LISTEN. THINK. SOLVE.®

INTEGRATED PRODUCTION & PERFORMANCE SUITE

Data Management

FactoryTalk® Historian SE

UNIINT INTERFACE USER MANUAL

PUBLICATION HSEUNI-UM001A-EN-E–July 2012

Allen-Bradley • Rockwell Software

Rockwell Automation

Contact Rockwell Automation
Customer Support Telephone — 1.440.646.3434
Online Support — http://www.rockwellautomation.com/support

Copyright Notice
© 2012 Rockwell Automation Technologies, Inc. All rights reserved. Printed in USA.
© 2010 OSIsoft, Inc. All rights reserved.

This document and any accompanying Rockwell Software products are copyrighted by Rockwell Automation Technologies, Inc. Any reproduction and/or distribution without prior written consent from Rockwell Automation Technologies, Inc. is strictly prohibited. Please refer to the license agreement for details.

Trademark Notices
FactoryTalk, Rockwell Automation, Rockwell Software, the Rockwell Software logo are registered trademarks of Rockwell Automation, Inc.

The following logos and products are trademarks of Rockwell Automation, Inc.:
FactoryTalk Historian Site Edition (SE), FactoryTalk Historian Machine Edition (ME), RSView, FactoryTalk View, RSView Studio, FactoryTalk ViewStudio, RSView Machine Edition, RSView ME Station, RSLinx Enterprise, FactoryTalk Services Platform, FactoryTalk Live Data, and FactoryTalk VantagePoint.

The following logos and products are trademarks of OSIsoft, Inc.:
PI System, Sequencia, Sigmafine, gRecipe, sRecipe, and RLINK.

Other Trademarks
ActiveX, Microsoft, Microsoft Access, SQL Server, Visual Basic, Visual C++, Visual SourceSafe, Windows, Windows ME, Windows NT, Windows 2000, Windows Server 2003, and Windows XP are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
Adobe, Acrobat, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

ControlNet is a registered trademark of ControlNet International.

DeviceNet is a trademark of the Open DeviceNet Vendor Association, Inc. (ODVA).

Ethernet is a registered trademark of Digital Equipment Corporation, Intel, and Xerox Corporation.

OLE for Process Control (OPC) is a registered trademark of the OPC Foundation.

Oracle, SQL*Net, and SQL*Plus are registered trademarks of Oracle Corporation.

All other trademarks are the property of their respective holders and are hereby acknowledged.

Restricted Rights Legend
Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Warranty
This product is warranted in accordance with the product license. The product's performance may be affected by system configuration, the application being performed, operator control, maintenance, and other related factors. Rockwell Automation is not responsible for these intervening factors. The instructions in this document do not cover all the details or variations in the equipment, procedure, or process described, nor do they provide directions for meeting every possible contingency during installation, operation, or maintenance.
This product's implementation may vary among users.
This document is current as of the time of release of the product; however, the accompanying software may have changed since the release. Rockwell Automation, Inc. reserves the right to change any information contained in this document or the software at anytime without prior notice. It is your responsibility to obtain the most current information available from Rockwell when installing or using this product.

Table of Contents

Introduction **1**

Related Rockwell Automation Documents 1

Interface Startup Parameters **3**

Startup Options 3
Conventions and Limitations 3
Running Modes 4
Informational Parameters 4
Standard Parameters 5

Historian Point Configuration **15**

What are Historian Points? 15
Interface Points 15
Point Attributes 16
Historian Tag String Attribute Limits 21
Input Points 22
Output Points 23
I/O Rate Points 25
Monitoring I/O Rates on Historian Interface Node 25
Configuring the Historian Point on Historian Server Node 25
Configuration on Historian Interface Node 25
Performance Points 26
Performance Summaries 27
UniInt and PI SDK 27
“Known Servers” Table 28

Interface Operation **29**

Libraries 29
Connection Establishment and Connection Recovery to Historian 29
UniInt Version 3.1.6 and Lower 29
UniInt Version 3.2.0 and Above 29
Historian Point Updates 30
Exit Handler 30
Time 31
UniInt 2.x Compatibility Mode 31

Windows Performance Counters 33

Requirements 33
Installing and Removing Interface Performance Counters 33
Verifying Successful Performance Counter Operation 34
Viewing Performance Counter Data 35
Historizing Performance Counter Data 35
UniInt Performance Counters 35
Changed Counters (Interface Upgrades) 36
Troubleshooting Performance Counter Problems 37

Interface Health Points 39

Extended Descriptor Requirements 41
General Interface Health Tags 42
Interface Point Count 43
IO Rate 43
Message Counter 44
Output Rate 44
Output Bad Value Rate 44
Trigger Input Rate 45
Trigger Input Bad Value Rate 45
Scan Class Health Tags 45
Scan Class Scan Time 48
Scan Class Input Device Scan Time 48

Interface Disconnected Startup 49

Disconnected Startup Requirements 49
Operating Systems 49
Interface 50
Buffering 50
Point Synchronization and Historian Server Connections 51
Point Modifications on the Historian Server and Interface Restarts 51
Limitations 52
Extended Attribute Lengths 52
Interfaces Using the PI SDK 52
Outputs 53
Startup Command File Configuration 54
Sample Interface Startup Files 55
ICU Configuration 55
Disconnected Startup Architecture 59
Operation 60
Creation 61
Validation 62
Construction 62
Utilization 63
Synchronization 64

Remote Cache Builds....65
Messages....66
Informational....66
Configuration Errors....68
Troubleshooting....70
Interface Operation....70
Deleting Cache Files....70

UniInt Failover Scheme....73

Introduction....73
Configuring UniInt Failover....75
Start-Up Parameters....75
Data Source Points....76
Historian Tags....77
Steady State Operation....81
Glossary....82
Active ID Point....82
Backup Interface....82
Cold Failover....82
Data Source....82
Failover ID....83
Failover Update Interval....83
Heartbeat Point....83
Hot Failover....83
Input Data....84
Interface Control Points....84
Interface Control Tags....84
Interface Copy....84
Interface Instance....84
Output Data....84
Primary Interface....85
Point85
Thrashing....85
Warm Failover....85
Failover Scenarios....85
Primary Interface Shutdown....85
Primary Interface Stops Updating Heartbeat Point....88
Manual Failover....90
Primary Interface Loses Connection to Historian Server....92
Primary Stops Updating Heartbeat Point for less than 5 Intervals....94
Interfaces Connect to Data Source at Approximately the Same Time....96
Maximum Data Overlap....97

Error and Informational Messages.....99

Message Logs 99
Messages on Windows 99
System Errors and PI Errors 99
Error Descriptions on Windows 100

Introduction

UniInt is an abbreviation for Universal Interface. UniInt is a framework for interfaces to the Historian Server. UniInt provides generic functions required by most interfaces such as establishing a connection to the Historian Server node and monitoring the Historian Point Database for changes. Using the UniInt framework ensures a standard set of features for Rockwell Automation interfaces to Historian and prevents duplication of effort. Using the UniInt framework also makes it possible to add new functionality such as interface level failover and enhanced interface monitoring to a large majority of Rockwell Automation's standard interfaces to the FactoryTalk Historian System.

This manual explains the operation and functionality that is provided by the UniInt framework. Not all of the features described in this manual are supported by every interface that is built on top of the UniInt framework. Please check the interface specific manual for a description of the features supported.

Please see the Interface specific manual for details on installing and configuring each individual interface.

Related Rockwell Automation Documents

The following table describes documents that are related to interface operation. The document is applicable only if the interface is running on the platform that is listed in the table.

Document	Description
UniInt Interface User Manual (Uniint Interface User Manual.pdf)	This document. This manual describes common features of UniInt-based interfaces.
Interface-specific documentation	Each UniInt-based interface has interface-specific documentation that describes its features, functions, and implementation.

Document	Description
Release notes for UniInt (UniIntReleaseNotes.doc)	Release notes contain specific information about a particular version of UniInt. Release notes typically include: • System requirements • Installation notes • Upgrading • Whats new • Changes in this release • Outstanding issues • Known limitations • Contact information
PI API Installation Instructions (API_Install.doc)	This manual contains the installation instructions for the PI API and Bufserv. If the interface is installed on a Windows node and if the Historian Server is installed on a different node, then the interface node is commonly referred to as a Historian Interface node. The PI API must be installed on a Historian Interface node to enable communication between the interface and the Historian Server. Bufserv is a utility program that stores and forwards events to a Historian Server, allowing continuous data collection when the server is down for maintenance, upgrades, backups, and unexpected failures.

Interface Startup Parameters

Startup parameters determine the behavior of an interface. This chapter documents the available startup parameters for a UniInt based interface. Consult the interface specific documentation to discover which parameters are and are not supported by a particular interface.

Startup Options

Interface startup parameters can be specified using any one of the following three methods:

- Manually enter a series of parameters on the command-line.
- Create a startup file that contains a series of parameters. Issue a command to run the startup file.
- Create a startup file that contains a series of parameters. Start a service that runs the command file.

Conventions and Limitations

A Windows command file name must have a **.bat** extension.

There is no limit to the number of parameters in a startup command file. The maximum line length in Windows is 1024 characters (1 kilobyte). The maximum length for a parameter in Windows is also 1 kilobyte.

Continuation characters allow specifying parameters over multiple lines. The continuation character for Windows is **^** (up arrow caret).

Command-line parameters can begin with a **/** (forward slash) or with a **-** (dash). For example, **/ps=M** is equivalent to **-ps=M**. Command-line parameters that contain spaces must be enclosed in double quotes (“). For example, **/path=C:\pipc\interfaces** is a valid parameter but **/path=C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC\interfaces** will not be parsed correctly because of the space in the path. In order for the path parameter to be parsed correctly, either the value or both the parameter and value must be surrounded by double quotes. For example, both of the following are valid parameters: **"/path=C:\Program Files\Rockwell**

Software\FactoryTalk Historian\PIPC\interfaces" and **/path="C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC\interfaces"**.

Each entry in the following tables presents the parameter and its description. The Parameter column includes the UniInt default value for the parameter if there is one and whether or not the parameter is required or optional.

Note: Parameter requirements vary from interface to interface. Refer to the interface-specific documentation for interface-specific parameter requirements and for the exceptions for using standard UniInt parameters.

Running Modes

Interfaces typically run as detached or background processes until stopped by a manual shutdown or a system shutdown. There are a few parameters that cause an interface to output some information and then shutdown immediately. An example of one of these parameters is the **/help** parameter. If a UniInt based interface is executed with the **/help** parameter, the interface will start, notice the parameter **/help** request, output the help informational messages and then shut down. There is a short list of these parameters in a separate table labeled Informational Parameters because all of the parameters in this list will cause a UniInt based interface to start, output some information and then shut down. These command-line parameters must appear on the command-line by themselves.

Informational Parameters

Parameter	Description
/? Optional	On Windows, /? is the same as the /help parameter. On all other platforms /? is the same as /h.
/h Optional	Print a summary of command-line options supported by UniInt. The interface-specific command-line options will also be printed if this is supported by the interface.
/help Optional, Windows Only	In addition to printing the summary of command-line options from the /h parameter, print a summary of command-line options for installing, removing, and starting a Windows service.
/v Optional	Print the version of the interface, the PI API installed on the interface computer and the version of UniInt the interface was built against.

Standard Parameters

These command-line parameters are used in the every-day running of an interface. This table does not contain the entire list of parameters supported by UniInt. There are parameters that are specific to configuring disconnected start-up discussed in the Disconnect Start up section (page 54) and parameters specific to configuring UniInt Failover discussed in Configuring UniInt Failover section (page75). None of the parameters are case sensitive. For example, /ps=ifc is equivalent to /ps=IFC.

Parameter	Description
`/AppName=Name` **Optional** **Default = 1st four characters of the interface executable**	The `/AppName=Name` controls the name sent to the Historian Server in order to establish the PI API trust. Prior to version 4.1 of UniInt, the interface name was used to establish the PI API trust with the Historian Server. This limited the Historian Administrator's options when setting up security for an interface node. There was no way to configure different PI API trusts for different instances of the interface running on the same computer. The Historian Administrator is now able to specify an application name for each instance by using the `/AppName=Name` parameter. The maximum length of the `Name` is 4 characters. Please keep in mind when setting up the Trust that the PI API puts an E at the end of the application name. The Application Name that is sent to the Historian Server can be viewed in the Historian Server Message Log immediately after an interface is started. A message will be written to the message log stating the connection status for a "process name". The "process name" is the application name sent from the interface.
`/dbUniInt=level` Optional Defualt = 0	The `/dbUniInt` parameter is used to set a debug level for debugging UniInt code. `level` is a 32-bit integer. Setting a particular bit in `level` to 1 turns on a particular debug switch. To turn on every debug switch specify: `/dbUniInt=0xFFFF` `level` can be specified as either a decimal number or a hexadecimal number. A `0x` must precede hexadecimal numbers. The following table lists the sections of code that can provide debug information and the hexadecimal value that `level` must equal in order for that section to output messages. Multiple sections can output debug messages simultaneously by adding the values together. For example, both initiailization and shutdown can be debugged by adding the two `level` Values 0x02 and 0x08 together and specifying `level` as 0x0A.

UniInt Area	Hexadecimal Value
Initialization	0x0002
Shutdown	0x0008
Saving to Historian	0x0010
Control Loop Function	0x0020

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>Point Loading</td><td>0x0040</td></tr>
<tr><td>Time Functions</td><td>0x0400</td></tr>
<tr><td>Digital State Caching</td><td>0x2000</td></tr>
</table>
</td></tr>
<tr><td>/DisableCounters

Optional</td><td>Windows Performance counters are enabled by default. Counters can be disabled by specifying /disablecounters on the command-line.</td></tr>
<tr><td>/ec
or
/ec=#

Optional</td><td>The first instance of the /ec parameter on the command-line is used to specify a counter number, #, for an I/O Rate point. If # is not specified, then the default event counter is 1. Also, if the /ec parameter is not specified at all, there is still a default event counter of 1 associated with the interface. If there is an I/O Rate point that is associated with an event counter of 1, each copy of the interface that is running without /ec=# explicitly defined will write to the same I/O Rate point. This means that one should either explicitly define an event counter other than 1 for each copy of the interface or one should not associate any I/O Rate points with event counter 1. Configuration of I/O Rate points is discussed in the section I/O Rate Points (page 25)

For interfaces that run on Windows nodes, subsequent instances of the /ec parameter may be used by specific interfaces to keep track of various input or output operations. One must consult the interface-specific documentation to see whether subsequent instances of the /ec parameter have any effect. Subsequent instances of the /ec parameter can be of the form /ec*, where * is any ASCII character sequence. For example, /ecinput=10, /ecoutput=11, and /ec=12 are legitimate choices for the second, third, and fourth event counter strings.</td></tr>
<tr><td>/f=SS
or
/f=SS,SS
or
/f=HH:MM:SS
or
/f=HH:MM:SS,hh:mm:ss

Required for reading scan-based inputs</td><td>The /f parameter defines the time period between scans in terms of hours (HH), minutes (MM), and seconds (SS). The scans can be scheduled to occur at discrete moments in time with an optional time offset specified in terms of hours (hh), minutes (mm), and seconds (ss). If HH and MM are omitted, then the time period that is specified is assumed to be in seconds.

Each instance of the /f parameter on the command-line defines a scan class for the interface. There is no limit to the number of scan classes that can be defined. The first occurrence of the /f parameter on the command-line defines the first scan class of the interface; the second occurrence defines the second scan class, and so on. Historian Points are associated with a particular scan class via the Location4 Historian Point attribute. For example, all Historian Points that have Location4 set to 1 will receive input values at the frequency defined by the first scan class. Similarly, all points that have Location4 set to 2 will receive input values at the frequency specified by the second scan class, and so on.

Two scan classes are defined in the following example:
/f=00:01:00,00:00:05 /f=00:00:07
or, equivalently:
/f=60,5 /f=7
The first scan class has a scanning frequency of 1 minute with</td></tr>
</table>

Parameter	Description

	an offset of 5 seconds, and the second scan class has a scanning frequency of 7 seconds with no offset.
	
	When no offset is specified, the scan class will be scheduled for immediate execution. That is, the interface will not wait for a well-defined moment in time before scanning when no offset is specified.
	
	When an offset is specified, the scans occur at well-defined moments in time according to the formula:
	
	scan times = (reference time) + n(frequency) + offset
	
	where n is an integer and the "reference time" is midnight on the day that the interface was started. In the above example, "frequency" is 60 seconds and "offset" is 5 seconds for the first scan class. This means that if the interface was started at 05:06:06, the first scan would be at 05:07:05, the second scan would be at 05:08:05, and so on.
	
	There is no guarantee that a scan will occur at the interval defined by the scan class. If the interface is under a large load, then some scans may occur late or be skipped entirely. See the section called "Performance Summaries" for more information on skipped or missed scans, page 27.
	
	Sub-second Scan Classes
	
	One can also specify sub-second scan classes on the command-line such as
	
	`/f=0.5 /f=0.1`
	
	where the scanning frequency associated with the first scan class is 0.5 seconds and the scanning frequency associated with the second scan class is 0.1 seconds.
	
	Similarly, sub-second scan classes with sub-second offsets can be defined, such as
	
	`/f=0.5,0.2 /f=1,0`
	
	Wall Clock Scheduling
	
	Wall-clock scheduling means that the scans will occur at the same time every day, even across daylight savings time transitions. The interface implicitly assumes that certain scan classes should be scheduled according to a wall clock. For example, the interface implicitly assumes that `/f=24:00:00,08:00:00` means that the scan should occur once a day at 8 AM. The period of the scan is actually 25 hours during daylight saving time to standard time transitions and is 23 hours during standard time to daylight savings time transitions.
	
	Wall-clock scheduling is assumed only when an offset is explicitly defined for a scan class. For example, `/f=24:00:00` means that the scan should occur once every 24 hours even across daylight savings time transitions. Even when an offset is explicitly defined, wall-clock scheduling is implicitly assumed only if the scan would ordinarily occur at the same time every day. For example, the scans associated with `/f=23:00:00,08:00:00` would occur at 8 AM the first day, 7 AM the next day, and so on. The interface will perform the scan once every 23 hours even across daylight savings time transitions since there is no implicit reason to

Parameter	Description
	adhere to a wall clock for this particular scan class. Periods of 2, 3, 4, 6, 8, 12, or 24 hours or periods that are a multiple of 24 hours are implicitly scanned according to wall-clock scheduling as long as a time offset is explicitly defined for the scan class. A scan class can be explicitly defined to adhere to wall-clock scheduling by appending, L to the scan class definition. For example, `/f=23:00:00,08:00:00,L` will invoke wall-clock scheduling for the scan class. This means that the period of the scan class will be 24 hours during daylight saving time to standard time transitions and 22 hours during standard time to daylight savings time transitions. Appending, L has no effect when the period of the scan class is less than or equal to 1 hour. Similarly, a scan class can be explicitly defined to ignore wall clock scheduling by appending, `U` to the scan class definition. For example, `/f=24:00:00,08:00:00,U` means that the scan period will always be 24 hours, even across daylight savings time transitions. An interesting case occurs with wall-clock scheduling on the spring forward time change for a scan class defined as `/f=02:00:00,00:30:00`. Under normal operation, scans will occur at 00:00:30, 02:30:00, 04:30:00, and so on. However, during the spring forward time change, there is no wall clock time of 02:30:00. In this case, the interface skips the scan entirely and performs the following scan at 04:30:00.

`/foxutc`
 	
`Optional`
	If this parameter is specified, UniInt will determine the UTC time of the Historian Server node in the same manner that the Foxboro interface determines the UTC time of the Historian Server node.
	
	This option has been added because there are situations where the default method of calculating the UTC offset between the interface node and the Historian Server node does not work properly. The Foxboro interface on Windows is one example where the default method does not work (whether or not the PI SDK is enabled). For Foxboro, the time zone on the interface node is always set to GMT standard time, but the wall clock time is still adjusted by 1 hour during daylight savings time changes. This means that the UTC time on the interface node jumps by 1 hour over a DST transition.
	
	When the PI SDK is not enabled, another example where the default method for calculating the UTC offset fails is when the interface is communicating to a Historian Server across a time zone that differs by a fraction of an hour. For example, the default method fails when there is a 30-minute difference in time zones between the interface node and the Historian Server node. The 30-minute timezone difference can be handled by enabling the /foxutc option or the /pisdk=1 option.
	
	This parameter is obsolete if the API version is 1.6.x or higher and the Historian Server is version 2.x or higher. If the API functions introduced with version 1.6.1 of the API and 2 of the Historian Server are supported, UniInt will use the API function pitm_fastservertimeutc to determine the UTC time of the Historian Server.

Parameter	Description
/host=host:port Optional **Default = default Historian Server from the pilogin.ini file.**	The /host parameter is used to specify the Historian Home node. host is the IP address of the Historian Sever node or the domain name of the Historian Server node. port is the port number for TCP/IP communication. It is recommended to explicitly define the host and port on the command-line with the /host parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults. Defaults: The default port name and server name is specified in the pilogin.ini or piclient.ini file. The piclient.ini file is ignored if a pilogin.ini file is found. Refer to the *PI API Installation Instructions* manual for more information on the piclient.ini and pilogin.ini files. If the interface is configured to use the PI SDK, localhost should not be used, rather the name of the IF node computer should be used. Examples: The interface is running on a Historian Interface node, the domain name of the Historian 3 home node is Marvin, and the IP address of Marvin is 206.79.198.30. Valid /host parameters would be: /host=marvin /host=marvin:5450 /host=206.79.198.30 /host=206.79.198.30:5450
/id=x Optional Default = none	The /id parameter is used to specify the interface identifier. For example, /id=int1 The interface identifier is a string that is no longer than 9 characters in length. UniInt concatenates this string to the header that is used to identify error messages as belonging to a particular interface. See the section called “Error and Informational Messages” for more information, page 99. UniInt always uses the /id parameter in the fashion described above. Many interfaces also use the /id parameter to identify a particular interface copy number that corresponds to an integer value that is assigned to one of the Location code point attributes, most frequently Location1. For these interfaces, one should use only numeric characters in the identifier. For example, /id=1 When an interface uses the /id parameter to specify an interface copy number, the point source character should be the same for all copies of the interface. For example, all Historian Points for the Modbus interface are typically configured with a point source of M. Modbus points are distinguished as belonging to a particular copy of the Modbus interface by assigning a copy number to the Location1 attribute of the point, which, in turn, corresponds to the copy number designated by the /id parameter. If the interface does not have a parameter such as the /id parameter to identify a particular copy of the interface, a different point source must be used for each copy of the interface. Interfaces sometimes use other parameters besides

Parameter	Description
	the /id parameter (such as /in, /ic, etc) to identify the copy of an interface. Consult the interface-specific documentation for more information.
/IOSourceTime Optional	The /IOSourceTime stands for Initial output source timestamp. Some interfaces send the current value of the source tag at start-up. If this parameter is not specified, the timestamp sent with the initial value is the current Historian Time. This command-line parameter instructs UniInt to send the timestamp of the source tag's snapshot value to the interface specific code that handles the actual output.
/logps Optional	When the /logps parameter is present, the point source for the interface will be included in the header of the log messages sent to the pipc.log file.
/logUFOID Optional	When the /logUFOID parameter is present and the interface is configured for UniInt failover, the failover ID for the interface will be included in the header of the log messages sent to the pipc.log file.
/maxstoptime= stoptime Optional Default = 120 seconds	When an interface receives a signal from the operating system to shut down, there is a list of clean up functions that need to be performed. If for some reason the execution of these functions takes longer than the stoptime, the interface will shut down without finishing its tasks.
/NoOutputs Optional	If this parameter is specified, outputs are disabled. If outputs are disabled, the message /NoOutputs flag detected. Outputs from Historian disabled will be written to the message log when the interface starts.
/perf=interval Optional Default = 8 hours	When the percentage of scans that a UniInt-based interface performs on time drops below 95%, UniInt will print out performance summaries for each scan class. Performance summaries are described in the section called "Performance Summaries." UniInt checks whether a performance summary is in order every interval hours. For example, if /perf=0.025, UniInt will write performance summaries every 90 seconds if the percentage of on-time scans is below 95%. The minimum time between summaries is 60 seconds. Setting /perf=0 disables summaries. If the /perf parameter is omitted, then UniInt checks whether summaries are in order every 8 hours by default. If the inputs for the interface are unsolicited, then performance summaries should be disabled by setting /perf=0 because performance summaries are meaningless for unsolicited input points.

Parameter	Description
/PISDK=# Optional Default = 0	The /pisdk parameter can be used to enable or disable the PI SDK in some situations. Use /pisdk=1 to enable the PI SDK. Use /pisdk=0 to disable the PI SDK. If a particular interface requires the PI SDK, then the PI SDK will always be enabled and the /pisdk parameter will be ignored. If the interface is running on an interface node with the PI API version 1.6.x or greater and the version of the Historian Server is 2.x or greater, the interface will ignore the /pisdk parameter and the SDK will not be used to retrieve point attributes.
/pisdkcontimeout= contimeout Optional Default = connection timeout set with the Connection Manager	contimeout is the timeout in seconds used for opening the initial connection to the Historian Server. All other PI SDK calls use the timeout defined by the /pisdktimeout parameter. The timeout set with this parameter overrides the connection timeout listed in the connection settings dialog that can be viewed from the Connection Manager. The Connection Manager should be used for adjusting the SDK connection timeout. This parameter should only be specified if the interface uses the PI SDK and a connection timeout that is different than the connection timeout set with the Connection Manager is required.
/pisdktimeout= timeout Optional Default = data access timeout set with the Connection Manager	timeout is the timeout in seconds used by PI SDK calls. The timeout set with this parameter overrides the data access timeout listed in the connection settings dialog that can be viewed from the Connection Manager. The Connection Manager should be used for adjusting the SDK data access timeout. This parameter should only be specified if the interface uses the PI SDK and a data access timeout that is different than the one set with the Connection Manager is required.
/ps=source Required	The /ps parameter specifies the point source for the interface. Source is not case sensitive. The length of source is limitied to 100 characters by UniInt. Source can contain any character except '*' and '?'. The point source that is assigned with the /ps parameter corresponds to the PointSource attribute of individual Historian Points. The interface will attempt to load only those Historian Points with the appropriate point source. If the PI API version being used is prior to 1.6.x OR the Historian Server version is prior to 2.x, source is limited to a single character unless the SDK is being used. See the /pisdk parameter for more information. Strategies for assigning a point source character vary depending on the interpretation of the /id parameter by a particular interface. See the /id parameter for more information.
/q Optional Default = no queuing	When the /q parameter is present, Snapshots and exceptions are queued before they are sent to the Historian Server node. The maximum queue size is close to 4000 bytes. The queue is flushed between scans if it is not filled. For interface collecting unsolicited data, the queue is flushed four times a second if it is not filled.

Parameter	Description
/SVCEventsTO=# Optional Default = 500 msec	The /SvcEventsTO=x controls the Service Events TimeOut. The x is the timeout period specified in milliseconds. Events are serviced (retrieved from the evmexceptions queue on the Historian Server) for 500 milliseconds or until there are no more events in the queue. After either of those events, UniInt will perform the other tasks it is responsible for such as scanning for input data and checking for Historian point database changes. If the interface is servicing a lot of events, the time may not be sufficient and can be adjusted with the SvcEventsTO=x command-line parameter. The minimum value for x is 0. If the Service Events TimeOut parameter is set to 0, UniInt will service 36 events at a time and then continue to perform its other tasks. The maximum value for x is 3000 (3 seconds).
/sio Optional Default = send initial outputs	The /sio parameter stands for "suppress initial outputs." The parameter applies only to interfaces that support outputs. If the /sio parameter is not specified, the interface will behave in the following manner. When the interface is started, the interface determines the current Snapshot value of each output tag. Next, the interface writes this value to each output tag. In addition, whenever an individual output tag is edited while the interface is running, the interface will write the current Snapshot value to the edited output tag. This behavior is suppressed if the /sio parameter is specified on the command-line. That is, outputs will not be written when the interface starts or when an output tag is edited. In other words, when the /sio parameter is specified, outputs will only be written when they are explicitly triggered.
/sn Optional Default = send exceptions	When this parameter is specified on the command-line, all exception reporting is disabled. Unless the interface-specific documentation says otherwise, this parameter should not be used during normal interface operation.
/startup_delay or /startup_delay=delay Optional Default = no delay Default if parameter is specified with no value = 30 seconds	After UniInt prints out the "Starting Interface." startup message, UniInt will wait delay seconds before proceeding. If the /startup_delay parameter is specified without specifying a delay time, then the default delay is 30 seconds.

Parameter	Description
/stopstat=digstate or /stopstat /stopstat only is equivalent to /stopstat="Intf Shut" Optional Default = no digital state written at shutdown.	If the /stopstat parameter is present on the startup command-line, then the digital state Intf Shut will be written to each Historian Point when the interface is stopped. If /stopstat=digstate is present on the command-line, then the digital state, digstate, will be written to each Historian Point when the interface is stopped. UniInt uses the first occurrence in the table. If neither /stopstat nor /stopstat=digstate is specified on the command-line, then no digital states will be written when the interface is shut down. Examples: /stopstat=shutdown /stopstat="Intf Shut" The entire digstate is enclosed within double quotes when there is a space in digstate.
/uht_id=# Optional Default = Health tags are not filtered by location3 unless UniInt failover is used	The /uht_id=# command-line parameter is used to specify a unique ID for interfaces that are run in a redundant mode without using the UniInt failover mechanism. There are several Rockwell Automation interfaces that are UniInt based and implement their own version of failover. In order for health tag(s) to be configured to monitor a single copy of the interface, an additional parameter is required. If the /uht_id=# is specified, only health tags with a location3 value = # will be loaded.
/uiinfops=s Optional (not recommended) Default = UI_IF_INFO	The /uiinfops command-line parameter is used in conjuction with the Interface Informaiton point and is described in more detail in the Interface Monitoring Points section. The interface by default looks for a point with the point source UI_IF_INFO and checks to see if the point is the Interface Monitoring point. This point source can be changed but if it is, the /uiinfops will have to be added to every interface in order for the interfaces to find the Interface Information point.
/uitrace=x Optional Default = no tracing enabled	If the /uitrace command-line parameter is specified, the values and timestamps that are received by the interface are also written to the log file. Tracing can be enabled for a particular point by specifying /uitrace=x, where x is the point number or tag name of the point to be traced. This parameter should be used with caution because specifying this parameter could cause the log file to become very large.

Parameter	Description
/updateinterval=# Optional Default = 120 seconds	This parameter can be used to adjust the interval with which UniInt checks for point updates. The default interval is 120 seconds, the minimum interval for checking for point updates is 1 second, and the maximum interval is 300 seconds. See the section on Point Updates for more information. This parameter can be set to 0. Setting the update interval to 0 will disable checking for point updates. Modifications to the Historian point database will not be discovered by the interface without stopping and then restarting the interface. If the Historian point database is very stable, this disabling the checking for point updates will slightly reduce the chances of missing scans when communication between the Historian Server and the interface is interrupted. An interface may miss scans at this time because the API function to check for point updates will not return until a response is received from the Historian Server or a timeout is reached. UniInt will not be scanning while it waits for the API call to return.

Historian Point Configuration

What are Historian Points?

The Historian point is the basic building block for controlling data flow to and from the Historian Data Archive. Whereas a point in geometry has coordinates associated with it that define its position in space, a Historian point has attributes associated with it that determine the source and destination of data, the means by which the data is transmitted, how the data is stored in Historian, etc. A single point is configured for each measurement value that needs to be archived. For example, one point can be configured to store temperature readings from a thermocouple. Another point could be configured to store temperature readings from a second thermocouple.

The purpose of most UniInt based interfaces is to move data from a device to Historian and, to a lesser extend, to move data from Historian to a device. Rockwell Automation publishes more than 400 standard interfaces to Historian so the list of devices is long on diverse. Each interface uses specific Historian point parameters for configuring data collection on a per point basis. The interface manual will have to be consulted for information on a specific interface. This section will list the attributes that are common to most interfaces. Each interface is different so check the interface manual to ensure the interface does not deviate from the standard use of these attributes.

Points for UniInt-based interfaces can be split into two broad classes: interface points and UniInt points. Interface points are the main means of controlling data transfer for an interface. As the name implies, the configuration of interface points depends upon the interface in question. UniInt points are used exclusively by UniInt for moving standard data such as I/O Rates, performance and health monitoring information, or control points such as Failover Control points (UniInt failover is discussed starting on page 71).

Interface Points

Interface points can be subdivided into input points and output points. A Historian point is called an **input point** when the flow of data is from some external source to the Historian Data Archive specified with the **/host** command-line parameter. A Historian point is called an **output point** when the flow of data is from the Historian Data Archive to a destination that is external to the Archive. The mechanism for determining which points are input points and which points are output points is specific to each interface. Please consult the interface

user's manual for detailed discriptions of the Historian Point Attributes that are relavant to the interface.

The following section describes common point attributes. The second and third sections describe how input and output points are configured and used.

Point Attributes

The following Historian Point Attributes are used by UniInt based interfaces. Refer to the interface documentation to determine whether the standard UniInt implementation for a particular attribute is applicable. Some attributes may have additional or different meanings for individual interfaces. If the specific attribute is not mentioned by the interface documentation of a UniInt based interface, then the attribute will affect the data in accordance with the explanation in this manual. Also, most UniInt based interfaces use a number of attributes that are not described here.

Tag

The words point and tag are frequently used interchangeably, but there is a difference. A tag is a label or name for a point. Any tag name can be used in accordance with the normal Historian point naming conventions.

PointSource

The PointSource is a unique, single or multi-character string that is used to identify the Historian point as a point that belongs to a particular interface. For example, a single point source using the letter **`R`** may be chosen to identify points that belong to the Random interface. In order to implement this, the PointSource attribute should be set to **`R`** for every Historian Point that is configured for the Random interface. Then, if **`/ps=R`** is specified on the startup-command-line of the Random interface, the Random interface will search the Historian Point Database for every Historian point that is configured with a PointSource of **`R`**. Likewise, a multi-character PointSource using the string **`Boiler1`** may be used to identify points that belong to the *PIIFC* Interface. To implement this, the PointSource attribute would be set to **`Boiler1`** for every Historian Point that is configured for the *PIIFC* Interface. Then, if **`/ps=Boiler1`** is used on the startup command-line of the *PIIFC* Interface, the Interface will search the Historian Point Database for every Historian point that is configured with a PointSource of **`Boiler1`**. Before an interface loads a point, the interface usually performs further checks by examining additional Historian Point Attributes to determine whether a particular point is valid for the interface. For additional information, see the **`/ps`** parameter.

The following table explains length limits imposed on the PointSource for different configurations. When the maximum possible lengths differ for the software installed on site, the shortest length applies.

PI API	Historian Server	PI SDK	Maximum Length
1.6 or higher	2.x or higher	na	100

PI API	Historian Server	PI SDK	Maximum Length
na	Below 2.x	Disabled	1
Below 1.6	na	Disabled	1
na	na	Enabled	100

Case-sensitivity for PointSource Attributes

In all cases, the point source character that is supplied with the **/ps** command-line parameter is not case sensitive. That is, **/ps=P** and **/ps=p** are equivalent.

Reserved and Illegal Point Sources

Several subsystems and applications that ship with Historian 3 are associated with default point source characters The Totalizer Subsystem uses the point source character **T**, the Alarm Subsystem uses **G** and **@**, and the Performance Equations Subsystem uses **C**. Rockwell Automation recommends not using these point source characters. Also, if a point source character is not specified when creating a Historian point, the point is assigned a default point source character of Lab. Therefore, it would be confusing to use **L** or **Lab** as the point source character for an interface.

The ***** and the **?** characters are invalid point sources as they are wild card characters for searching the database. These characters cannot be any part of a multi-character point source.

PointType

Consult the interface documentation to determine which point types are supported for individual interfaces. Not all point types that are supported by the Server are necessarily supported by an interface.

Typically, DCS point types do not need to correspond to Historian point types but this may be a requirement for certain interfaces. For example, integer values from a DCS can be sent to floating point or digital Historian tags. Similarly, a floating-point value from the DCS can be sent to integer or digital Historian tags, although the values will be truncated. This rule does not extend to string tags. UniInt will not convert an integer, float, or digital type value to a string. Unless the interface documentation specifies these conversions, only string and blob type data will be saved to a string Historian tag.

Note: If an interface is not using the extended PI API functionality, negative integer values from the DCS must be stored in floating point tags in Historian because negative integers cannot be transmitted by the traditional PI API. If using the extended PI API, int16 and digital tags are the only Historian point types with this limitation.

ExcMin, ExcMax, ExcDev and ExcDevPercent

These are the exception specification attributes.

The exception specification attributes are used to define exception-reporting specifications. Exception reporting is basically a filter. Raw values that the interface receives are filtered if they do not pass the exception-reporting specifications. If the raw values pass exception, they are sent to the Snapshot, where they can be viewed by client applications such as FactoryTalk Historian ProcessBook. Unless the **/sn** parameter is specified on the command-line of the interface, standard exception reporting will be implemented. The interface documentation must be consulted for exceptions to the rule.

ExcMin is the exception minimum time; ExcMax is the exception maximum time; and ExcDev is the exception deviation dead band.

ExcDev is defined in terms of engineering units. ExcDev is related to ExcDevPercent by:

```
ExcDev = ExcDevPercent * Span / 100
```

where Span is defined by the Span Historian point attribute. If either ExcDev or ExcDevPercent is changed, the other is automatically updated to be compatible. If both are changed at once, ExcDevPercent takes precedence.

Raw values are said to "pass exception" if:

- The difference between the new value and the last value that passed exception is greater than ExcDev
  and
  the difference between the times of the new value and the last value that passed exception is greater than ExcMin.

or

- The difference between the times of the new value and the last value that passed exception is greater than ExcMax.

The last value that passed exception is called the "old value." The next value that passes exception is called the "new value." The last value that was received by the interface before the new value is called the "previous value." There will not be a "previous value." if the interface did not receive a value between the "old value" and the "new value." When a new value passes exception, the previous value and the new value will be sent to the Snapshot. The new value will then become the old value, and the cycle continues.

The time between exceptions can be greater than the ExcMax if no new values are received by the interface for a point.

Note: The "previous value" will be sent to Historian even if it was received before ExcMin seconds had expired. ExcMin applies only to the "new value."

Compressing

If the compressing attribute is set to 1 (ON), then the following compression specifications are used to filter the data that is passed from the Snapshot to the Archive.

CompDev, CompMin, CompMax, CompDevPercent

These are the compression specifications. CompDev is the compression deviation; CompMin is the compression minimum time; and CompMax is the compression maximum time. CompDevPercent and CompDev are related by:

```
CompDev = CompDevPercent * Span / 100
```

where Span is defined by the Span Historian point attribute.

The "swinging door compression" algorithm is used to filter the data. The algorithm is described in the *Historian Server Reference Guide* manual.

Scan

By default, the Scan attribute has a value of 1, which means that scanning is turned on for the point. Setting the scan attribute to 0 turns scanning off. If the scan attribute is 0 when the interface starts, a message is written to the pipc.log and the tag is not loaded by the interface. There is one exception to the previous statement.

If any Historian Point is removed from the interface while the interface is running (including setting the scan attribute to 0), SCAN OFF will be written to the Historian Point regardless of the value of the Scan attribute. Two examples of actions that would remove a Historian Point from an interface are to change the point source or set the scan attribute to 0. If an interface specific attribute is changed that causes the tag to be rejected by the interface, SCAN OFF will be written to the Historian point.

Location4

For interfaces that support scan-based collection of data, Location4 defines the scan class for the Historian point. The scan class determines the frequency at which input points are scanned for new values. For more information, see the description of the **`/f`** parameter in the Interface Startup Parameters section beginning page 3.

Zero and Span Attributes

Zero and Span attributes are required for Historian point types of Int16 and Float16 because the values of Zero and Span attributes affect the data representation in the Archive for these point types. Zero and Span attributes for other point types may have an affect on the data collection especially if the compression and exception specifications are specified using the percentage point attributes. For more information, see *Historian Server Reference Guide*.

TypicalValue

Although the TypicalValue is only used to document a reasonable value for the Historian point, it must be specified if its default value will not fall between the Zero and Zero+Span. Otherwise, creation of the Historian Point will fail. For a numeric tag, it must be greater than

or equal to the Zero attribute and less than or equal to the Zero attribute plus the Span attribute.

The TypicalValue must be between Zero and Zero plus Span. For digital points, Historian sets the Zero and Span internally. See the description of Zero and Span attributes for more information.

SourceTag

The SourceTag attribute is used to designate a trigger tag for output points. The mechanism by which a SourceTag is used to trigger outputs is discussed under the section called "Output Points," page 23.

Shutdown

The shutdown attribute is used only if the server node is a Historian 3 system.

The Shutdown attribute is 1 (true) by default. The default behavior of the Historian Shutdown subsystem is to write the SHUTDOWN digital state to all Historian Points when Historian is started. The timestamp that is used for the SHUTDOWN events is retrieved from a file that is updated by the Snapshot Subsystem. The timestamp is usually updated every 15 minutes, which means that the timestamp for the SHUTDOWN events will be accurate to within 15 minutes in the event of a power failure. For additional information on shutdown events, refer to the *Historian Buffer Server.chm* file.

Note: The SHUTDOWN events that are written by the Historian Shutdown subsystem are independent of the SHUTDOWN events that are written by the interface when the `/stopstat` command-line parameter is specified.

SHUTDOWN events being written to Historian can be disabled when Historian is restarted by setting the Shutdown attribute to 0 for each point. Alternatively, the default behavior of the Historian Shutdown Subsystem can be changed to write SHUTDOWN events only for Historian Points that have their Shutdown attribute set to 0. To change the default behavior, edit the **`\PI\dat\Shutdown.dat`** file, as discussed in *Historian Server Reference Guide.*

It is undesirable to write shutdown events when buffering is being used. Bufserv and PIBufSS are utility programs that provide the capability to store and forward events to a Historian Server, allowing continuous data collection when the Server is down for maintenance, upgrades, backups, and unexpected failures. That is, when Historian is shutdown, Bufserv will continue to collect data for the interface, making it undesirable to write SHUTDOWN events to the Historian Points for this interface. Disabling Shutdown is recommended when sending data to a Highly Available Historian Server Collective. Refer to the Bufserv manual for additional information.

ExDesc

This is the extended descriptor attribute. The extended descriptor is a string attribute. Please refer to the Historian Tag String Attribute Limits section beginning on page 21 for a description of the maximum number of characters available to the interface for different configurations. Typically, if the extended descriptor is used by an interface, it is used to identify a point, address or location on the data source. If an interface uses the extended descriptor for this purpose, the interface manual will have a detailed description of the requirements for this attribute. UniInt uses the extended descriptor to identify Performance Points (page 26), Trigger Tags (page 22), Failover Control Tags (page 71) and Interface Health points (page 25).

InstrumentTag

The instrument tag is a string attribute. Please refer to the Historian Tag String Attribute Limits section beginning on page 21 for a description of the maximum number of characters available to the interface for different configurations. Typically the instrument tag is used to identify a point, address or location on the data source. If an interface uses the instrument tag for this purpose, the interface manual will have a detailed description of the requirements for this attribute.

DigitalSet

The DigitalSet attribute is used to assign a digital state set to a Historian Point. For standard usage, refer to *Historian Server Reference Guide.*

Historian Tag String Attribute Limits

UniInt retrieves three string type attributes from the Historian Server during point loading other than the point source which is described above. The three attributes are the tag name (Tag), extended descriptor (ExDesc) and the instrument tag (InstrumentTag). The instrument tag is not used by UniInt, but the string value for this attribute is sent to the interface during point loading. The extended descriptor is checked by UniInt for a number of key words that may identify the point as a UniInt point. The extended descriptor is also passed to the interface during point loading. The tag name is only used for logging error and debugging messages. UniInt saves a maximum of 21 characters for the tag name after point loading is completed. If the tag name is longer than 21 characters, UniInt will save the first ten characters and the last ten characters of the tag name separated by a tilde (~). If the tag name length exceeds the number of characters UniInt can retrieve, the last ten characters will be the last ten UniInt is able to retrieve during point loading. The number of characters UniInt can retrieve for each attribute depends on the PI API version, the Historian Server version and whether or not the PI SDK is being used to retrieve point attributes. The following table describes the number of characters UniInt can retrieve given the configuration. The first column of the table is “new API functions”. The new API functions that support retrieving the longer attributes are only available if the PI API is version 1.6.x or higher AND the Historian Server is version 2.x or higher. Please consult the interface documentation to determine if the interface requires UniInt to load the PI SDK. If the –PISDK startup parameter is not specified by the interface, the value is 0.

New API functions	Interface Requires SDK	/PISDK Startup Parameter	Max Tag Name	Max Instrument Tag	Max Extended Descriptor
no	no	0	254	32	83
no	no	1	1023	1023	1023
no	yes	na	1023	1023	1023
yes	na	na	1023	1023	1023

Input Points

Input points are used to write data to the Historian Data Archive. Each interface has its own rules for determining whether a given point is an input point or an output point. There is no *de facto* Historian point attribute that distinguishes a point as an input point or an output point.

Input points can be scan-based, trigger-based, or unsolicited.

Scan-based Inputs

For scan-based input points values are sent to the points on a periodic basis. Scanning periods are defined by the **`/f`** parameter on the startup-command-line of the interface. The first appearance of the **`/f`** parameter on the command-line is referred to as the first scan class, the second appearance of the **`/f`** parameter is referred to as the second scan class, and so on.

The Location4 Historian point attribute is used to associate an input point with a given scan class. For example, to associate a particular input point with scan class 1, set Location4 to 1 for the point.

Trigger-based Inputs

For trigger-based input points, a separate trigger point must be configured. An input point is associated with a trigger point by entering a case-insensitive string in the extended descriptor (ExDesc) Historian point attribute of the input point of the form:

```
keyword=trigger_tag_name
```

where **`keyword`** is replaced by “event”, “trig” or “trigger” and **`trigger_tag_name`** is replaced by the name of the trigger point. UniInt automatically assumes that an input point is trigger-based instead of scan-based when the **`keyword=trigger_tag_name`** string is found in the extended descriptor attribute.

An input is triggered when a new value is sent to the Snapshot of the trigger point. The new value does not need to be different than the previous Snapshot value to trigger an input, but the timestamp of the new value must be greater than (more recent than) or equal to the

timestamp of the previous value. This is different than the trigger mechanism for output points. For output points, the timestamp of the trigger value must be greater than (not greater than or equal to) the timestamp of the previous value.

As of UniInt 3.3.4, conditions can be placed on trigger events. Event conditions are specified in the extended descriptor as follows:

```
Event='trigger_tag_name' event_condition
```

The trigger tag name must be in single quotes. For example,

```
Event='Sinuoid' Anychange
```

will trigger on any event to the Historian Tag sinusoid as long as the next event is different than the last event. The initial event is read from the snapshot. Depending on the interface, developer-specific event conditions may be configurable. The tag name must be surrounded by the single quote and there can be no text between the second or closing single quote and the trigger keyword.

The keywords in the following table can be used to specify trigger conditions.

Event Condition	Description
Anychange	Trigger on any change as long as the value of the current event is different than the value of the previous event. System digital states also trigger events. For example, an event will be triggered on a value change from 0 to "Bad Input," and an event will be triggered on a value change from "Bad Input" to 0.
Increment	Trigger on any increase in value. System digital states do not trigger events. For example, an event will be triggered on a value change from 0 to 1, but an event will not be triggered on a value change from "Pt Created" to 0. Likewise, an event will not be triggered on a value change from 0 to "Bad Input."
Decrement	Trigger on any decrease in value. System digital states do not trigger events. For example, an event will be triggered on a value change from 1 to 0, but an event will not be triggered on a value change from "Pt Created" to 0. Likewise, an event will not be triggered on a value change from 0 to "Bad Input."
Nonzero	Trigger on any non-zero value. Events are not triggered when a system digital state is written to the trigger tag. For example, an event is triggered on a value change from "Pt Created" to 1, but an is not triggered on a value change from 1 to "bad input."

Unsolicited Inputs

Some interfaces do not poll for data. Rather, the foreign device (Distributed Control System, PLC, SCADA, etc.) is in control of what type of data is sent to the interface and when it is sent. For unsolicited data input, the value of Location4 is meaningless to UniInt. The interface may make use of Location4 for some other Historian tag configuration value. Please see the interface documentation for a description of how Location4 is used for unsolicited input tags.

Output Points

Output points control the flow of data from the Historian Data Archive to any destination that is external to the Historian Data Archive, such as a PLC or a third-party database. For

example, to write a value to a register in a PLC, one would use an output point. Each interface has its own rules for determining whether a given point is an input point or an output point. There is no *de facto* Historian point attribute that distinguishes a point as an input point or an output point.

Outputs are triggered for UniInt-based interfaces. That is, outputs are typically not scheduled to occur on a periodic basis. There are two mechanisms for triggering an output.

As of UniInt 3.3.4, event conditions can be placed on triggered outputs. The conditions are specified using the same event condition keywords in the extended descriptor as described under "Trigger-Based Inputs." The only difference is that the trigger tag is specified with the SourceTag attribute instead of with the "event" or "trig" keywords. Otherwise, the behavior of event conditions described under 'Trigger-Based Inputs" is identical for output points. For output points, event conditions are specified in the extended descriptor as follows:

```
event_condition
```

Trigger Method 1 (Recommended)

For trigger method 1, a separate trigger point must be configured. The output point must have the same point source as the interface. The trigger point can be associated with any point source, including the point source of the interface. Also, the point type of the trigger point does not need to be the same as the point type of the output point.

The output point is associated with the trigger point by setting the SourceTag attribute of the output point equal to the tag name of the trigger point. An output is triggered when a new value is sent to the Snapshot of the trigger point. The new value does not need to be different than the previous value that was sent to the Snapshot to trigger an output, but the timestamp of the new value needs to be more recent than the previous value. If no error is indicated, then the value that was sent to the trigger point is also written to the output point with the same timestamp as the trigger point. If the output is unsuccessful, then an appropriate digital state that is indicative of the failure is usually written to the output point. If an error is not indicated, the output still may not have succeeded because the interface may not be able to tell with certainty that an output has failed.

Trigger Method 2

For trigger method 2, a separate trigger point is not configured. To trigger an output, write a new value to the Snapshot of the output point itself. The new value does not need to be different than the previous value to trigger an output, but the timestamp of the new value must be more recent than the previous value.

Trigger method 2 may be easier to configure than trigger method 1, but trigger method 2 has a significant disadvantage. If the output is unsuccessful, there is no tag to receive a digital state that is indicative of the failure, which is very important for troubleshooting.

I/O Rate Points

I/O Rate points are used to monitor the data collection rate of an interface. An I/O Rate point can be configured to receive 10-minute averages of the total number of exceptions per minute that are sent to Historian by the interface. A value is considered an exception if it has exceeded the exception limits for a given Historian point.. Since 10-minute averages are taken, the first average is not written to Historian until 10 minutes after the interface has started. One I/O Rate tag can be configured for each copy of the interface that is in use.

Monitoring I/O Rates on Historian Interface Node

For Windows nodes, the 10-minute rate averages (in events/minute) can be monitored with a client application such as ProcessBook. The IOMonitor program is discussed on page DA-71 of *FactoryTalk Historian System Manual I*.

Configuring the Historian Point on Historian Server Node

Create an I/O Rate Tag with the following point attribute values.

Attribute	Value
PointSource	L
PointType	float32
Compressing	0
ExcDev	0

The default settings can be used for the remaining Historian Point Attributes.

When Compressing is set to Zero the I/O Rate Tag acts like a heartbeat tag for the interface, which can be examined easily in FactoryTalk Historian ProcessBook with markers turned on. If a value is not written to the I/O Rate Tag every 10 minutes, there is a problem with the interface communication. The I/O Rate tag provides a quick diagnosis of the health of the interface, since flatlining beyond a 10 minute period or a rapid decrease in the event rate could indicate a communication problem that requires further analysis."

Configuration on Historian Interface Node

It is recommended to use the ICU to create the `iorates.dat` file over creating this file manually. User desiring to manual create the `iorates.dat` file should use the following procedure. For the following examples, assume that the name of the Historian tag is `inf001`, and that the name of the I/O Rate on the home node is `inf001`.

Windows Nodes

1. Edit/Create a file called `iorates.dat` in the **`PIHOME\dat`** directory. The **`PIHOME`** directory is defined either by the **`PIPCSHARE`** entry or the **`PIHOME`** entry in the

pipc.ini file, which is located in the **%windir%** directory. If both are specified, the **PIPCSHARE** entry takes precedence.

Since the PIHOME directory is typically **C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC**, the full name of the **iorates.dat** file will typically be **C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC\dat\iorates.dat**.

Add a line in the iorates.dat file of the form:

inf001, #

where **inf001** is the name of the I/O Rate Tag and **#** corresponds to the first instance of the **/ec=#** parameter in the startup command file. **#** can be any number between 1 and 34 or between 51 and 200, inclusive. However, it is best to use an event counter, **#**, that is not equal to 1 because 1 is the default event counter for UniInt-based interfaces. To specify additional rate counters for additional copies of the interface, create additional I/O Rate tags and additional entries in the **iorates.dat** file. The event counter, **/ec=#**, should be unique for each copy of the interface.

2. Set the **/ec=#** parameter on the startup command file of the interface to match the event counter in the **iorates.dat** file.

3. The interface must be stopped and restarted in order for the I/O Rate tag to take effect. I/O Rates will not be written to the tag until 10 minutes after the interface is started. A message in the **pipc.log** file will indicate if the I/O Rate tag was properly configured and loaded.

Performance Points

Performance points can be configured to monitor the amount of time in seconds that it takes an interface to complete a scan for a particular scan class. The closer the scan time is to 0 seconds, the better the performance. The scan time is recorded to millisecond resolution. Several other measurements concerning the performance of the interface can be monitored with performance points..

Performance point configuration is the same on all operating system platforms. Performance points are configured as follows.

1. Set the extended descriptor to:

   PERFORMANCE_POINT

   or to:

   PERFORMANCE_POINT=interface_id

   where **interface_id** corresponds to the identifier that is specified with the **/id** parameter on the startup command-line of the interface. The character string **PERFORMANCE_POINT** is case insenstive. The **interface_id** does not need to be

specified if there is only one copy of an interface that is associated with a particular point source.

2. Set Location4 to correspond to the scan class whose performance is to be monitored. For example, to monitor scan class 2, set Location4 to 2. See the **/f** parameter for a description of scan classes.

3. Set the PointSource attribute to correspond to the **/ps** parameter on the startup command-line of the interface.

4. Set the PointType attribute to any of the floating-point point types that are supported by the Historian Server.

Performance Summaries

UniInt monitors interface performance by keeping track of the number of scans that are hit, missed, and/or skipped for scan-based input points. Scans that occur on time are considered hit. If a scan occurs more than 1 second after its scheduled time, the scan is considered missed. If a scan occurs 1 scan period or more after its scheduled time, then 1 or more scans are considered skipped. Say that a particular scan class has a period of 2 seconds. If a scan for this class occurs 1.1 seconds after its scheduled time, then 1 scan has been missed. However, no scans have been skipped because the next scan still has the opportunity to occur at its scheduled time, which happens to be 0.9 seconds after the last scan in this case. For scans that have periods of 1 second or less, the above definition of a missed scan does not make sense. In these cases, scans are considered either hit or skipped. Since every skipped scan is also considered to be a missed scan, the scan performance summary should indicate the same percentage of skipped and missed scans for scan classes with periods of 1 second or less.

By default, UniInt prints out a performance summary to the message log every 8 hours if the hit ratio (hit ratio = hits / (hits + misses)) drops below 0.95. The performance summary shows the percentage of scans that are missed and skipped for every scan class. The frequency at which performance summaries are printed out can be adjusted using the **/perf** command-line parameter.

For interfaces that use unsolicited input points, performance summaries should be inactivated by setting **/perf=0** because performance summaries are meaningless for unsolicited inputs.

UniInt and PI SDK

Prior to the release of the version 1.6.0.2 of the PI API, the PI SDK was required in order to retrieve long extended descriptors, instrument tags and tag names. The SDK was also required in order to a use multi-character point source. With the release of the PI API 1.6.0.2, UniInt based interfaces can support all of these features without using the PI SDK. UniInt version 4.1 was modified to take advantage of the new API making the use of the PI SDK

For Windows-based interfaces, PI SDK support is available beginning with version 3.2.0 of UniInt. UniInt uses the PI SDK to support the following features:

- Extended descriptors and instrument tags up to 1 kilobyte in length, or 1023 characters (UniInt 3.2.0). Starting with version 4.1.0 of UniInt, if version 1.6.x or greater of the PI API is installed on the interface node and the Historian Server is version 2.x or greater, UniInt will use the PI API functions that were released with PI API 1.6.x to get instrument tags and extended descriptors of up to 1023 characters.
- Full 32-bit values for excmin and excmax (UniInt 3.4.4).
- Multi-character point source characters (UniInt 3.5.0). Starting with version 4.1.0 of UniInt, if version 1.6.x or greater of the PI API is installed on the interface node and the Historian Server is version 2.x or greater, UniInt will use the PI API function for retrieving points with a multi-character point source.

With the PI API prior to version 1.6, the extended descriptor was limited to 80 characters, the instrument tag was limited to 32 characters, excmin and excmax were limited to 16-bit values (0 – 65,535), and only single-character point sources were supported.

Check the documentation for your particular interface to see whether the PI SDK is enabled or can be enabled. If the PI SDK is enabled, the following message will be written to the log file.

```
The PI SDK is enabled
```

If the above message is printed, the version of the PI SDK being used will also be printed. If the PI SDK is not enabled, the following message will be written to the log file.

```
The PI SDK is disabled
```

"Known Servers" Table

In order for the PI SDK to be able to connect to a particular Historian Server, the Historian Server must be configured in the PI SDK's "Known Servers" table. The "Known Servers" table can be edited using the **`\pipc\pisdk\AboutPI SDK.exe`** utility. Since the PI API uses the **`pilogin.ini`** file for its Server list, Server information must be configured in two different spots for PI API / PI SDK hybrid applications.

Interface Operation

Libraries

UniInt 3.1.6 and lower versions use the PI API instead of the PI SDK. The PI API is described in detail in the *FactoryTalk Historian application Programming Interface* manual. The PI API library is distributed with the PI SDK. For installation instructions, refer to the *PI API Installation Instructions* manual.

Version 1.3.x of the PI API is required for UniInt 3.x.

UniInt version 3.2.0 and above use both the PI API and the PI SDK. However, the PI SDK libraries are used only if the developer specifically compiles UniInt to use the PI SDK or if the user configures the PI SDK with the **`/PISDK=1`** command-line parameter. There is an exception to the **`/PISDK=1`** parameter. If the interface developer does not inform UniInt at compile time that the PI SDK is required AND if the version of the PI API is 1.6 or higher AND if the Historian Server is version 2 or higher, the **`/PISDK=1`** switch is ignored because all of the functionality that was previously unavailable to the PI API is now available.

Connection Establishment and Connection Recovery to Historian

UniInt Version 3.1.6 and Lower

UniInt establishes the initial connection to Historian and reconnects to Historian in the event that the connection is lost for some reason. If the interface is started while the Historian Server is down, the interface will try to establish a connection until the Historian Server is up.

UniInt Version 3.2.0 and Above

The following applies if the UniInt was compiled to use its PI SDK features. Otherwise, the description in the section “UniInt 3.1.6 and Lower” applies.

When UniInt is compiled to use the PI SDK, UniInt must establish a connection to Historian via both the PI API and the PI SDK. A connection via both the PI API and the PI SDK must

be established before the interface will continue. If the interface is started while the Historian Server is down, the interface will try to establish the connections until the Historian Server is up.

Historian Point Updates

When a UniInt-based interface starts, the interface searches the Historian Point Database for points that belong to the interface (see the PointSource attribute under "Historian Point Configuration") and a point list is created for the interface. Once startup is complete, UniInt checks the Historian Point Database every 2 minutes for points that are added, edited, and deleted. If point updates are detected, the points are loaded (or reloaded) by the interface as appropriate. The 2 minute update interval can be adjusted with the /updateinterval command-line parameter.

UniInt will only process 25 point updates at a time. If more than 25 points are added, edited, or deleted at one time, UniInt will process the first 25 points, wait 30 seconds (or by the time specified by the **`/updateinterval`** parameter, whichever is lower), process the next 25 points, and so on. Once all points have been processed, the interface will resume checking for updates every 2 minutes (or by the time specified by the /updateinterval parameter).

UniInt will write the digital state **`SCAN OFF`** to any points that are removed from the interface while the interface is running.

Exit Handler

UniInt's exit handler performs the following operations in the listed order.

1. UniInt calls the interface-specific exit handler to perform interface-specific cleanup operations.

2. If the **`/stopstat=digstate`** parameter is present on the command-line of the interface, UniInt will write the **`digstate`**, digital state, to all scan-based and event-based input points of the interface. If a performance point is associated with the interface, **`digstate`** will also be written to the performance point. I/O Rate points do not receive **`digstate`** events because 0 is written to I/O Rate points when the interface exits. See the description of the **`/stopstat`** parameter under the section called "The Startup Command File" for recommended usage of the **`/stopstat`** parameter,

3. If an I/O Rate point is associated with the interface, the I/O Rate point will be updated with a final rate, and the I/O Rate counter will be zeroed.

It is possible for the interface to be terminated without invoking the exit handler. One normally does not want to bypass the exit handler. On any operating system, the exit handler will not be invoked when the computer crashes. For interactive processes on Windows, the exit handler will be terminated prematurely if control-c is hit a second time (the exit handler is invoked the first time that control-c is hit).

Time

UniInt 3.x handles time differently depending upon whether or not UniInt is running in extended PI API mode. A startup message in the `pipc.log` file clearly indicates whether or not UniInt is running in extended PI API mode. When the extended PI API features of UniInt are not enabled, UniInt 3.x handles time in the same manner as UniInt 2.x. This is called UniInt 2.x compatibility mode.

UniInt 2.x Compatibility Mode

UniInt 2.x compatibility mode means that the extended PI API features of UniInt are not enabled. In this mode, UniInt-based interfaces typically timestamp data using one of the following:

1. The system time on the Historian Server node.

2. A timestamp provided by the DCS.

However, the interface-specific documentation must be consulted to determine the exact manner in which time is handled.

When the system time on the server node is used to timestamp the data, UniInt keeps track of a time offset between the server node and the interface node and updates this offset every 30 minutes. The local system time plus the offset is equal to the Historian Server time. If communication to the home node is lost, UniInt will apply the last offset that it calculated until the offset can be verified when communication to the server is restored. If the time offset between the server node and the interface node changes by more than 30 minutes, UniInt will not adjust the time offset because UniInt assumes that a mistake was made in adjusting the time either on the server or the interface node. An appropriate error message will be written to the `pipc.log` file.

Many interfaces that support the use of a DCS timestamp provide two methods for time stamping the data. One option is for the interface to use the DCS time directly. In this case, it is up to the user to make sure that the DCS system time is synchronized with the system time on the Historian Server node. At the very least, the DCS system time cannot be more than 10 minutes ahead of the system time on the Historian Server node because Historian cannot archive values that are more than 10 minutes in the future. A second option for the interface is to apply an offset to the DCS timestamp, where the offset is equal to the server time minus the DCS time. The offset is typically recalculated on a periodic basis.

Extended PI API Mode

In extended PI API Mode, UniInt interfaces typically use one of the following to timestamp data:

1. Coordinated Universal Time (UTC).

2. A timestamp provided by the DCS.

However, the interface specific documentation must be consulted to determine the exact manner in which time is handled.

UTC time is defined here as the number of seconds since January 1, 1970 00:00:00 in Greenwich, England. As of version 3.1.3, UniInt uses the UTC time of the Historian Server node instead of the UTC time of the local interface node. Ideally, the UTC times of the server and interface nodes are the same because UTC time is the same in every time zone. However, there is normally some drift between clock times.

If an interface supports DCS timestamps, the DCS timestamp must not be more than 10 minutes ahead of the Historian Server time because Historian cannot archive events that occur more than 10 minutes in the future.

Windows Performance Counters

Counters are used to provide information as to how well an application, service, or driver is performing. Developers can define counters for an interface. The total number of interface-specific and UniInt counters cannot exceed 200. This maximum is hard coded. Any counters in excess of 200 will be ignored.

Requirements

As of UniInt 3.5.0, UniInt always tries to enable Windows performance counters when the interface runs as a service as long as the **`/disablecounters`** command-line parameter is not specified. The following requirements must be met before counters will work.

1. The **`pictrdll.dll`** must be installed in the PIPC\bin directory in order for performance counters to work.

2. The **`piperfmondll.dll`** is installed with PI API 1.3.4 and greater in the PIPC\Interfaces\piperfmon_basic directory.

3. Counters are activated only if the interface is running as a Windows service. Counters are not activated if the interface is running interactively.

Installing and Removing Interface Performance Counters

Interface performance counters are installed automatically by the interface the first time that the interface is started as a Windows service. Since it is possible for other processes to uninstall performance counters for the interface, the interface checks to see if performance counters need to be re-installed every time that the interface is started as a service. Interface performance counters are removed when the interface service is removed with the **`-remove`** command-line option. The service can be removed even while the service is running. However, once the service is removed, the performance counters cannot be viewed via the Windows performance monitor unless the interface counters were already being viewed before the service was removed.

Interface-specific performance counters are installed in the registry under

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\`
`<service_name>+<ServiceID>\Performance`

where **service_name** is the interface executable name without the .exe extension and **ServiceID** is a service identifier that can be specified with the **-serviceid** command-line parameter when the service was installed. If the **-serviceid** command-line parameter is omitted when the service was installed, then the service identifier is NULL.

The variables under the **Performance** key should look similar to:

```
Close:REG_SZ:PIPrfData_Close
Collect:REG_SZ:PIPrfData_Collect
First Counter:REG_DWORD:0x82c
First Help:REG_DWORD:0x82d
Last Counter:REG_DWORD:0x83c
Last Help:REG_DWORD:0x83d
Library:REG_SZ:D:\PIPC\bin\pictrdll.dll
Open:REG_SZ:PIPrfData_Open
```

General performance counter information is stored in the registry under

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
\CurrentVersion\Perflib
```

The variables under the **Perflib** key should look similar to:

```
Base Index:REG_DWORD:0x737
ExtCounterTextLevel:REG_DWORD:0x1
LastCounter:REG_DWORD:0x83c
LastHelp:REG_DWORD:0x83d
Version:REG_DWORD:0x10001
```

The total number of counters that have been installed can be determined from the LastCounter variable under the **Perflib** key. In the examples above the LastCounter and LastHelp variables under the **Perflib** and **Performance** keys are the same, which means that the interface performance counters were the last counters to be installed.

Verifying Successful Performance Counter Operation

If performance counters are successfully activated, the following message with be written to the **pipc.log** when the interface starts:

```
Counters Successfully Activated (counters supported only for
Windows services only)
```

Otherwise, an appropriate error message will be written. Note that no messages will be written if the interface is started interactively because performance counters are supported only for Windows services.

While the interface is running as a service, an entry similar to

```
<No Name>:REG_SZ:<service_name>+<ServiceID>_Counters
```

will appear under the registry key

```
HKEY_LOCAL_MACHINE\SOFTWARE\PISystem\PI\Counters\<service_name>+<S
erviceID>\
```

When the interface service is stopped, the **<No Name>** entry under the registry key is removed. If the entry is not removed, this may indicate that the interface terminated abruptly without a chance to perform cleanup operations.

Viewing Performance Counter Data

The values that are written to the performance counters can be viewed with the Windows Performance Monitor. On Window, the performance monitor is launched from Start | Programs | Administrative Tools (Common) | Performance Monitor. To view counters from within the Windows Performance Monitor, do the following.

1. Select the *Add to chart* command from the Edit menu.
2. From the *Object* drop-down, choose the appropriate object as identified by the service display name of the interface.
3. From the *Counter* list box select the counter to be viewed. To get a detailed description of the selected counter, click the *Explain>>* button.
4. To add the selected counter, click the *Add* button.

Historizing Performance Counter Data

The data that are written to performance counters can be written to Historian with the Historian Performance Monitor Interface. There is a FULL version and a BASIC version of this interface. The basic version has four limitations:

- ❑ It must run on the same machine as the Historian Server.
- ❑ It will collect data for a maximum of 512 points.
- ❑ Only one instance of the interface is allowed.
- ❑ Data can be collected for performance counters only on the local machine.

The full version does not have these limitations. For additional information, please see the Historian Performance Monitor Interface Manual.

UniInt Performance Counters

In addition to interface-specific counters, the following performance counters are associated with UniInt-based interfaces for which counters are enabled.

Counter Name	Description
Interface up-time (seconds)	The number of seconds since the interface has started. This counter is incremented once a second.

Counter Name	Description
IO Rate (events/second)	The number of events per second received by the interface. If this counter is viewed from the Windows performance monitor, one should increase the update time of the performance monitor to the minimum scan period for the interface. For example, say that the minimum scanning period for the interface is 5 seconds (/f=5). One can set the update rate of the performance monitor to 5 seconds by selecting the "Chart…" command from the Options menu. In the dialog box, change the periodic update time to 5 seconds. If the update time is left at 1 second, then the IO Rate will appear to go to zero between scans because no events are received between scans.
Point Count	Number of points loaded by the interface.
Scan Time (milliseconds)	Time in milliseconds to call developer function and to write values to Historian. There is one "Scan Time" counter per scan class.
Scheduled Scans: % Missed	Percentage of missed scans since starting the interface. If a scan occurs more than 1 second after its scheduled time, the scan is considered missed. There is one "Missed Scans" counter per scan class. Related counters: Scheduled Scans: % Skipped Scheduled Scans: Scans this interval
Scheduled Scans, % Skipped	Percentage of skipped scans since starting the interface. If a scan occurs 1 scan period or more after its scheduled time, then 1 or more scans are considered skipped. There is one "Skipped Scans" counter per scan class. Related counters: Scheduled Scans, % Missed Scheduled Scans, Scans this interval
Scheduled Scans: Scans this interval	The total number of scans in the current performance interval. This total is the current sample size that is used to evaluate the % Missed Scans and the % Skipped scans. The performance interval is 8 hours by default, but this can be changed by the /perf command-line parameter. The minimum performance interval is 60 seconds (see the /perf parameter for more information). When the performance interval is exceeded, the previous samples are discarded and the sampling starts again from scratch. Related counters: Scheduled Scans, % Skipped Scheduled Scans, % Missed
Log file message count	Number of messages that have been written to the log file. Only applies to the instance _Total.
Points edited in the interface	Number of point edits that have occurred. Only applies to the instance _Total.
Points added to the interface	Number of point that have been added to the interface. Only applies to the instance _Total
Points removed from the interface	Number of point that have been removed from the interface. Only applies to the instance _Total

Changed Counters (Interface Upgrades)

UniInt checks whether performance counters are installed for the interface every time that the interface starts as a service. If performance counters are not installed, then performance

counters will be installed. If performance counters are already installed, then the total number of counters for the interface is checked. If the total number of counters has changed, then the interface counters are uninstalled and re-installed. Therefore, if the developer adds a performance counter in a new interface version, the new counter will automatically be installed the next time that the interface is run as a service.

However, if in a new interface version individual counters have been changed but the total number of counters is the same, the new counters will not automatically be updated the next time that the interface is run as a service. In order to load the new counters, the currently installed counters must be uninstalled by removing the service with the **`-remove`** command-line parameter. Once the service has been re-installed and the service has been re-started, the new counters will be installed.

Troubleshooting Performance Counter Problems

Error -30300, Unable to install performance counters for service test

This problem usually occurs when performance counters have been installed incorrectly by some application. The biggest offender is Network Associates VirusScan Windows version 4.03a and lower. If this version of VirusScan is installed, performance counters are most likely screwed up on your system. This can be verified by going to

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Windows\CurrentVersion\Perflib`

and checking the value of LastCounter. If LastCounter is not and even number, then performance counters are screwed up. One can work around this problem as follows:

1. Uninstall counters for the interface by removing the service.

2. Increment LastCounter and LastHelp under **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Windows\CurrentVersion\Perflib`** by 1.

3. Re-installing the interface as a Windows service.

4. Starting the interface as a service.

However, the above workaround may not be a permanent fix. If VirusScan is subsequently removed, counters will be screwed up again, and the workaround will need to be applied again. The only permanent way of fixing the counters is to re-install Windows and all software on your system.

Error Activating Counters, Could not find D:\PIPC\bin\pictrdll.dll

The interface will look for the **`pictrdll.dll`** in the directory specified by the **`Library`** variable under the registry key

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service_name>+<ServiceID>\Performance`

The **pictrdll.dll** should be installed with version 1.3.4 and greater of the PI API in the **PIPC\bin** directory.

Interface Health Points

There are several different types of points that may be used to monitor the health of an interface. The first type includes IO rate points, performance points, and Windows Performance counters. These points represent the traditional older methods for monitoring the health of an interface. IO rate and performance points are supported by UniInt based interfaces. The Windows performance counters require the Historian-PerfMon interface in order to get their values into Historian.

The second type of points is called Interface Health Points. The Interface Health points are configured in a manner very similar to performance points. UniInt looks at several interface startup parameters and Historian tag properties to determine if a point is to be loaded as an Interface Health point. The table below lists the applicable tag attributes, the required value and a description for each property. Any Historian Tag property not listed in the table is not applicable to the operation of Interface Health points.

Tag Attribute	Value	Description
PointSource	Equal to **/PS** from the interface startup parameters	The pointsource property for the tag is not case sensitive
Location1	Equal to **/ID** from the interface startup parameters	If there is no **/ID** found in the list of startup parameters, Location1 must be 0.
Location2	0	Not Used
Location3	Equal to **/UFO_ID** or **/UHT_ID** from the interface startup parameters	Only applicable if UniInt failover is being used or if the **/UHT_ID=#** is specified in the list of startup command parameters. A point with every other property set correctly will not be loaded by the interface if Location3 is not equal to the **/UFO_ID** parameter or **/UHT_ID** parameter.
Location4	Specifies the Scan Class to which this point pertains.	Only applicable to Scan Class points, all other Interface Health points ignore this value. For monitoring unsolicited IO Rate and Bad Value Rate, Total Scans Missed or Total Scans Skipped; Location4 must be 0
Exdesc	Must contain the proper Key Word described below	Key Word must be the first thing in the exdesc and is case sensitive.

The third type of points used to monitor interfaces is the Interface Information point. This point is different than any of the other points used for monitoring the health of an interface. There can be only one Interface Information point on a Historian Server and the pointsource does not match the **/ps** startup parameter for an interface.

UniInt supports collecting a variety of information about the health of an interface. The points that collect data pertaining to the performance of an interface are collectively referred to as Interface Health Points. The following paragraphs describe the supported Interface Health points, including configuration requirements and update intervals.

The non Scan Class Health points update either on change or at the heartbeat rate. The heartbeat rate is determined by the scan classes defined for the interface. If there are no scan classes defined, the heartbeat will update once per second. If there is at least one valid scan class defined for the interface, the heartbeat will update at the rate of the scan class with the highest frequency (smallest value). The other non Scan Class Health points will be updated when the heartbeat is updated. All of the Scan Class Health points will be updated when the class is scanned. For the Scan Class Health points, Location4 must be set to a valid scan class. It cannot be less than zero or greater than the number of valid scan classes specified in the list of startup parameters.

The following table lists the Interface Health Points, the keyword that must be specified in exdesc, the type of data written to the tag, when the point is updated and when it is reset to 0. The table is followed by a detailed description of the points. The rate at which the heartbeat tag is updated is dependant on the scan classes defined for the interface. Please see the detailed description of the heartbeat tag for the details on when the heartbeat tag is updated. Some of the other Health tags are updated when the heartbeat tag is updated. These tags are marked 'Heartbeat' in the updated column of the table. Some of the Health tags act as accumulators and keep a running total of events until they are reset to 0. When the reset occurs, the total is reset to a value of 0 and the accumulating of events start over. For a tag that is never reset to 0 until the interface is stopped and restarted, the tags will be marked as 'na' in the reset column. Several tags have their value reset to zero at the "Reporting Period". The reporting period for the tag is based on the performance summary interval. The default performance summary period is 8 hours. The performance summary period may be modified with the **/perf=*interval*** command-line parameter. During normal shutdown of the interface, the system digital state of "Intf Shut" will be written to all health tags loaded by the interface.

Health Point	Exdesc Keyword	Data Type	Updated	Reset
Heartbeat	[UI_HEARTBEAT]	Integer	Variable	na
Device Status	[UI_DEVSTAT]	String	On Change	na
Scan Class Information	[UI_SCINFO]	String	Perf Interval	na
Interface Point Count	[UI_POINTCOUNT]	Integer	On Change	na
IO Rate	[UI_IORATE]	Integer	Heartbeat	Heartbeat
Message Count	[UI_MSGCOUNT]	Integer	60 seconds	na
Output Rate	[UI_OUTPUTRATE]	Integer	Heartbeat	Reporting Period
Output Bad Value Rate	[UI_OUTPUTBVRATE]	Integer	Heartbeat	Reporting Period

Trigger Rate	[UI_TRIGGERRATE]	Integer	Heartbeat	Reporting Period
Trigger Bad Value Rate	[UI_TRIGGERBVRATE]	Integer	Heartbeat	Reporting Period
Scan Class IO Rate	[UI_SCIORATE]	Integer	At Scan	At Scan
Scan Class Bad Value Rate	[UI_SCBVRATE]	Integer	At Scan	At Scan
Scan Class Scan Count	[UI_SCSCANCOUNT]	Integer	At Scan	Reporting Period
Scan Class Scans Skipped	[UI_SCSKIPPED]	Integer	At Scan	Reporting Period
Scan Class Point Count	[UI_SCPOINTCOUNT]	Integer	At Scan	na
Scan Class Scan Time	[UI_SCINSCANTIME]	Integer	At Scan	na
Scan Class Device Scan Time	[UI_SCINDEVSCANTIME]	Integer	At Scan	na

Extended Descriptor Requirements

All of the keywords entered into the ExDesc must adhere to the following rules. The keywords must be the very first thing in the ExDesc with no spaces before the keyword, the keyword must be surrounded by square brackets ([ ]) with no spaces between the brackets and the keyword, and the keywords are all case sensitive. Additional text, such as a more detailed description of the tag, can be added to the ExDesc after the keyword if the Historian administrator would like a more detailed description or some other text that may help with the administration of these Historian tags. The additional text is not required, has no standard format and would serve no purpose other than helping a user or Historian administrator determine a tag's purpose from looking at the exdesc where the keywords are found. For example, the following is a valid extended descriptor for an IO Rate Health Point.

```
[UI_IORATE] Number of events per second for interface PIIFC
```

The following extended descriptors are examples of invalid keyword entries for an Interface Health point. If the keyword is not specified exactly, the interface will try to load the point as a regular interface point. This most likely will result in a message stating the point was not loaded by the interface. This may also result in the point being loaded by the interface and collecting unexpected data.

Invalid: no square brackets around the keyword

```
UI_IORATE Number of events per second for interface xyz
```

Invalid: spaces between the opening square bracket and the keyword

```
[ UI_IORATE] Events per second
```

Invalid: missing closing square bracket

```
[UI_IORATE
```

Invalid: keyword is in the wrong case

```
[ui_iorate] Number of events per second for interface xyz
```

General Interface Health Tags

The general interface health tags pertain to the operation of the interface in general. A list of scan class health tags follows. The scan class health tags require a valid scan class be configured in Location4 of the Historian tag. The general Interface Health tags do not consider the Location4 value.

Heartbeat

ExDesc Keyword: `[UI_HEARTBEAT]`

Update Interval: Once a second or based on scan class as described below.

Description:

The heartbeat tag is the primary tag used to determine if the interface is running. If the value of the heartbeat tag is updating, then the interface is running but not necessarily connected to and collecting data from a data source. The default update interval is 1 second if there are no scan classes defined for the interface. If the interface has defined scan classes, the update interval will be set to the highest frequency (lowest value) scan rate with the following limitations. If the scan rate is less than 1 second, the update interval will be set to 1 second. If the scan rate for the highest frequency scan class is greater than 1 minute, the update interval will be set to 60 seconds. The value written to the heartbeat tag increments from a value starting at 1, increments to a value of 15 and then restarts at 1.

The heartbeat tag will stop updating if the interface is shut down or if the interface is in a deadlock situation.

Note: The Interface Health Heartbeat tag is completely separate from the UniInt Failover Heartbeat tag. An interface can have both an Interface Health Heartbeat tag and a UniInt Failover Heartbeat tag loaded simultaneously.

Scan Class Information

ExDesc Keyword: `[UI_SCINFO]`

Update Interval: Once at interface startup and each performance summary interval.

Description:

The Scan Class Information Tag is a string tag that contains the number of scan classes used by the interface and their associated scan rates. The Heartbeat scan rate is also included in the string. The tag is updated once at interface startup and at each performance interval.

As an example, the value for the Scan Class Information tag for an interface with three scan class of 5 seconds, 10 seconds and 60 seconds will be “3 | 5 | 5 | 10 | 60”. The first number is the number of scan classes and the second number is the Heartbeat update rate. The numbers that follow the Heartbeat depend on the rate and number of scan classes defined. Each scan class rate will be written in seconds.

Device Status

ExDesc Keyword: `[UI_DEVSTAT]`

Update Interval: On startup, on change, and shutdown.

Description:

The device status tag stores communication information about the interface and the foreign device. This tag will be evaluated only while the heartbeat tag is updating. The device status tag is a string type Historian point. During normal shutdown of the interface, the string text of "4 | Intf Shutdown" will be written to this tag.

Format: n | uniint provided string text | interface specific text (when provided).

The 'n' value in the text and the pipe separator are formatting devices which may be used to quickly parse and analyze the value of the device status tag.

The update values and meaning of this tag are as follows.

- System Digital State "Good" – the interface is properly communicating with the foreign device and is able to read/write data to the device as required. When in this state, the system digital state of "Good" will be written to the Device Status tag.
- "1 | Starting" – the interface is starting.
- "2 | Connected / No Data" – the interface is connected to the foreign device. While in this state, the interface is not capable or reading or writing data to the foreign device. Optional text that may be provided by the interface is appended to the UniInt provided string value.
- "3 | *n* device(s) in error" – the interface is not able to communicate with *n* device(s). Optional text that may be provided by the interface is appended to the UniInt provided string value.
- "4 | Intf Shutdown" – the interface is shutting down.
- "5 | *interface_spcific_message*" – Text provided by the interface will be the only text following the "5 |".

Interface Point Count

ExDesc Keyword: `[UI_POINTCOUNT]`

Update Interval: On startup, on change, and shutdown.

Description:

Counts number of Historian tags loaded by the interface. This count includes all input, output and triggered input tags. This count does NOT include any Interface Health tags or performace points.

IO Rate

ExDesc Keyword: `[UI_IORATE]`

Update Interval: Based on heartbeat tag update.

Description:

Counts number of all values (inputs, outputs, triggered inputs) being sent to Historian before exception processing occurs. This tag will update based on the the update rate of the heartbeat tag and may report zero for update intervals where no data was sent to Historian. If the value stops updating in Historian then this is an indication that the interface has stopped collecting data.

Message Counter

ExDesc Keyword: `[UI_MSGCOUNT]`

Update Interval: Every 60 Seconds.

Description:

The Message Counter tag keeps track of the number of messages the interface has logged to the pipc.log file since interface start-up. The value of this tag is used solely for ad-hoc queries and diagnostics. It is a general rule-of-thumb that if the interface is logging a large number of messages, there is probably a problem that should be investigated.

Output Rate

ExDesc Keyword: `[UI_OUTPUTRATE]`

Update Interval: Based on heartbeat tag update interval and each performance summary interval.

Description:

Counts number of output tag events that are sent to Historian before exception processing. Output tags update based on some event and cannot be scheduled for an update rate. Therefore, the value of these tags will be updated at the same rate as the heartbeat tag and will reset after the performance summary interval.

Note: If no output points are defined, the digital state value of "**No Result**" will be written to the point.

Output Bad Value Rate

ExDesc Keyword: `[UI_OUTPUTBVRATE]`

Update Interval: Based on heartbeat tag update interval and each performance summary interval.

Description:

Counts number of output tag events that are sent to Historian before exception processing whose status is anything other than good. Output tags update based on some event and cannot be scheduled for an update rate. Therefore, the value of these tags will be updated at the same rate as the heartbeat tag and will reset after the performance summary interval.

Note: If no output points are defined, the digital state value of "**No Result**" will be written to the point.

Trigger Input Rate

ExDesc Keyword: `[UI_TRIGGERRATE]`

Update Interval: Based on heartbeat tag update interval and each performance summary interval.

Description:

Counts number of triggered input tag events that are sent to Historian before exception processing. Triggered input tags update based on some event and cannot be scheduled for an update rate. Therefore, the value of these tags will be updated at the same rate as the heartbeat tag and will reset after the performance summary interval.

Note: If no trigger input points are defined, the digital state value of "**No Result**" will be written to the point.

Trigger Input Bad Value Rate

ExDesc Keyword: `[UI_TRIGGERBVRATE]`

Update Interval: Based on heartbeat tag update interval and each performance summary interval.

Description:

Counts number of triggered input tag events that are sent to Historian before exception processing whose status is anything other than good. Triggered input tags update based on some event and cannot be scheduled for an update rate. Therefore, the value of these tags will be updated at the same rate as the heartbeat tag and will reset after the performance summary interval.

Note: If no trigger input points are defined, the digital state value of "**No Result**" will be written to the point.

Scan Class Health Tags

Unless otherwise stated in the description of the tag, a scan class health tag will receive the system digital state "No Result" if a scan class is defined and a scan class health tag for the class is loaded by the interface but there are no interface points loaded for the scan class. For

tags configured to monitor a particular scan class, the Historian tag attribute Location4 must be set to a valid scan class. Scan classes are defined with the **/f** startup parameter. Please see the list of startup parameters for a description of the **/f** startup parameter.

Scan Class Point Count

ExDesc Keyword: `[UI_SCPOINTCOUNT]`

Update Interval: When the associated class is scanned.

- ❑ If Location4 is 0, the point monitors the point count for unsolicited tags.
- ❑ If Location4 is greater than 0, Location4 specifies the scan class.

Description:

The Scan Class Point Count Tag is the number of tags loaded by the interface for a particular scan class. The value of this tag is updated after every scan. If there are no points loaded by the interface for the associated scan class, 0 will be written to the Scan Class Point Count tag.

Scan Class IO Rate

ExDesc Keyword: `[UI_SCIORATE]`

Update Interval: At the completion of associated scan class.

- ❑ If Location4 is 0, the point monitors the IO Rate for unsolicited tags.
- ❑ If Location4 is greater than 0, Location4 specifies the scan class.

Description:

The Scan Class IORate Tag is used to show the health of the input tags defined for a particular scan class. It is used to indicate the number of events sent to Historian before exception processing for the given scan class. As long as the current value of the tag is between zero and the scan class point count inclusive, then the scan class update executed successfully. If the Scan Class IORate Tag stops updating in Historian, this indicates an error has occurred and the tags for that scan class are no longer receiving new data. The value of this tag is updated after the completion of each scan.

Scan Class Bad Value Rate

ExDesc Keyword: `[UI_SCBVRATE]`

Update Interval: At the completion of associated scan class.

- ❑ If Location4 is 0, the point monitors the IO Rate for unsolicited tags.
- ❑ If Location4 is greater than 0, Location4 specifies the scan class.

Description:

The Scan Class Bad Value Rate Tag is used to calculate all the bad values being sent to Historian before exception processing from the device with a status of anything other than good. This tag can be defined for each scan class and is used for all input tags in that particular scan class.

Scan Class Scans Skipped

ExDesc Keyword: `[UI_SCSKIPPED]`

Update Interval: Event driven and each performance summary interval.

- If Location4 is 0, the point monitors the total scans skipped for all of the scan classes.
- If Location4 is greater than 0, Location4 specifies the scan class.

Description:

The Scan Class Scans Skipped tag keeps track of the number of scans that were not performed before the scan time elapsed and the next scheduled scan executed. The value of this tag is event driven and updates each time a scan is skipped. The value written to the Scan Skipped tag is the total number of scans skipped since the last reporting period. The value of the tag is set to zero at the beginning of each reporting period.

Scan Class Scan Count

ExDesc Keyword: `[UI_SCSCANCOUNT]`

Update Interval: At the completion of associated scan class.

- If Location4 is 0, the point monitors the total number of scans for all of the scan classes.
- If Location4 is greater than 0, Location4 specifies the scan class.

Description:

The Scan Class Scan Count tag keeps track of the number of scans that were performed by the interface since the last reporting period. The value of this tag is incremented after the completion of every scan. The value of the tag is set to zero at the beginning of each reporting period.

Scan Class Scan Time

ExDesc Keyword: `[UI_SCINSCANTIME]`

Update Interval: At the completion of associated scan class.

Description:

The Scan Class Scan Time is the total amount of elapsed time in milliseconds to read data from the foreign device, populate the input tags, and then send the data to Historian. The value for this tag is updated after each completed scan.

Scan Class Input Device Scan Time

ExDesc Keyword: `[UI_SCINDEVSCANTIME]`

Update Interval: At the completion of associated scan class.

Description:

The Scan Class Device Scan Time is the time in milliseconds to read data from the foreign device and populate the input tags. The Scan Class Device Scan Time is a subset of the total Scan Class Scan Time and can be used to determine the percentage of time spent communicating to the foreign system compared with the percentage of time spent communicating with Historian.

If the Scan Class Skipped tag value is increasing, the Scan Time tags along with the Device Scan Time tags can be used to help identify where the delay is occurring – the foreign system communication, the FactoryTalk Historian System communication, or elsewhere in the interface controlloop.

Interface Disconnected Startup

Interfaces built with UniInt version 4.3.0.x or later include functionality to take advantage of the Disconnected Startup operation. Simply stated, disconnected startup allows the interface to start from a local cache file with or without a valid connection to the host Historian Server. This critical functionality prevents data loss when an interface needs to start up and it does not have a connection to the Historian Server.

In addition, even if a connection to the host Historian Server is available, if the interface is configured for disconnected startup, the interface will utilize the local cache to quickly get the interface started and collecting data from the Data Source. Using the disconnected startup feature when a Historian Server connection is available bypasses potentially long delays caused by the latency in network calls required to gather point information from the Historian Server, which in turn delays the ability of the interface to begin data collection from the Data Source. As an example, an interface with 50,000 associated Historian Points may take several minutes to retrieve all the Historian Points from the Historian Server. Using the disconnected startup feature can reduce the interface time to load the same 50,000 Historian Points from the cache file to approximately 10 seconds. Results will vary depending on the system architecture and network latency. However, using disconnected startup will improve interface startup time, especially when large point counts are being used.

The first time the interface is started with the disconnected startup parameter, a connection to the Historian Server will be required to populate the cache files with the necessary data to support the disconnected startup feature on subsequent interface restarts. Two local cache files will be created on the interface node. The time to startup will be delayed while these files are being created. The local cache files contain the required Historian Server version, Historian point configuration, and Digital State information needed to start the interface. Once started, the interface can actively collect and buffer data retrieved from the Data Source when no connection to the host Historian Server is available.

Disconnected Startup Requirements

The minimum requirements necessary to run a UniInt based interface in a disconnected startup configuration are as follows:

Operating Systems

- Windows: Server 2003, XP Pro SP2, 2000 Server SP4

Interface

- The interface must be built against UniInt version 4.3.0.x or later.
- The interface node must have PI API version 1.6.1.5 or later installed.
- The interface must not utilize the PI SDK to retrieve the extended point attributes from the Historian Server.

Buffering

The interface node must have the Historian Buffering service installed and configured to buffer data to prevent data loss while starting the interface in the disconnected startup configuration when the host Historian Server is not available. Prior to the disconnected startup feature, the interface could not start without a connection to the Historian Server because the point configuration data needed to start the interface could only be retrieved from the Historian Server. However, with disconnected startup, the interface receives point configuration data from the cache files when no Historian Server connection is available. If the buffering service is not running while the Historian Server is unavailable, the benefit of being able to start the interface without a connection to the Historian Server is negated. Moreover, any data collected while the connection to the Historian Server is unavailable would be permanently lost.

There are two buffing services available; API Buffer Server and the Buffer Subsystem. API Node buffering (BufServ) refers to functionality in the PI API that supports continuous collection of data on an API Node regardless of the status of the destination Historian Server or the network link to the Historian Server. The Buffer Subsystem (PIBufss) is a new component of the FactoryTalk Historian System, primarily designed to enhance the High Availability (HA) features of the Historian Server. PIBufss is different from the API Buffer Server, even though it has most of the same capabilities and more.

The Buffer Subsystem offers significant benefits over the API Buffer Server, but does not replace it entirely. PIBufss has advantages over Bufserv when connected to redundant Historian Servers, configured as a Collective (Historian Server Replication) for High Availability (HA). These Historian Servers must be running Rockwell Automation Platform Release 1 (PR1) or a later version.

The PI API/BufServ 1.6.1.x introduced buffering to multiple Historian Servers from a single machine. This includes buffering to replicated Historian Servers that are part of a single Historian Collective. However, BufServ does not detect and validate the Historian Collective configuration and as a result, requires manual configuration. Additionally, due to the independent data compression taking place in the Snapshot Subsystem (at each replicated Historian Server), BufServ does not guarantee identical records in the Historian Archive.

Detailed documentation for configuring and running API Node Buffering can be found in the PI SDK Help (pisdk.chm, which embeds the piapi.chm file). The help file is located in the `PIHOME\PIPC\Help\` directory. The configuration details for the Buffer Subsystem can be found in the Buffer Subsystem Help (pibufss.chm file).

Point Synchronization and Historian Server Connections

An interfaces using the disconnected startup configuration has the ability to synchronize interface point configuration data with that of the host Historian Server. With interfaces built with older versions of UniInt that did not support disconnected startup, if the connection to the Historian Server was lost after interface startup, point modifications made on the Historian Server while the interface was disconnected were not retrievable by the interface once the connection to the Historian Server was restored. The interface needed to be restarted to get the latest point information.

With disconnected startup configured, point information for the interface will be synchronized with the Historian Server once a valid connection to the Historian Server becomes available. For example, an interface is started using the disconnected startup configuration. The connection to the Historian Server is lost due to a network failure or some other reason. While the interface connection to the Historian Server is down, the administrator modifies the point database. The connection to the Historian Server is now available. The interface now synchronizes its point cache to match the current point configuration on the Historian Server. Without restarting the interface, points that were created on the Historian Server while disconnected are added to the interface. Similarly, edited points are updated in the interface and deleted points are removed from the interface.

Note: Point changes made on the Historian Server while the interface is running but disconnected from the Historian Server will generate informational messages, in the pipc.log file, relating to point creation, editing, or deletion as appropriate. These messages will appear once per point modification. In some cases, an error message may be generated due to data being sent to Historian Points that no longer exist on the Historian Server. Typically, the interface only logs one message for a given error code only once per point that is in error. These informational and error messages are normal and should require no action by the user.

Point Modifications on the Historian Server and Interface Restarts

An interface started using the disconnected startup configuration loads points designated for the interface from locally stored cache files. This means that the interface only retrieves point information from the Historian Server during normal updates or during point synchronization. Therefore, changes made in Historian point database will show up in the interface as a typical point create, edit, or delete and users will notice pipc.log file messages showing point create, edit, or delete information relating to the modifications made to the Historian point database. The interface typically process its first point update 2 minutes after starting up and processes 30 point updates at a time. If more than 30 updates are pending, they will be processed at a rate of 30 updates every 30 seconds. Interfaces that experience numerous point changes will require additional time before the interface handles all the point changes.

It is recommended to run the interface in the non-disconnected startup mode while making major point modifications to the host Historian Server. Moreover, move or delete any previously created cache files for the interface to prevent the interface from using the outdated cache files during subsequent restarts when the disconnected startup feature is

enabled. Once satisfied with interface operation and data collection, enable disconnected startup as described in this document and restart the interface.

Limitations

The following section describes the known limitations imposed on an interface running in the disconnected startup configuration.

Extended Attribute Lengths

To maximize the functionality of the disconnected startup configuration, the use of a Historian Server version 2 or later is recommended. The use of Historian Server version 2 or later in conjunction with PI API version 1.6.1.x or later provides greater field lengths for the **`tag, descriptor`**, **`exdesc, instrumenttag`**, and **`pointsource`** attributes. Moreover, the PI SDK is no longer required for support of the increased field lengths. The following table lists the maximum field lengths for a given Historian Point attribute for the version of Historian Server used. In all cases, the PI API version is assumed to be 1.6.1.x or later to support disconnected startup.

Attribute Maximum Length (PI API 1.6.1.x or later on Interface Node)		
Attribute	Historian Server 2.x or higher	Historian Server Below 2.x
Tag	**1023**	**255**
Descriptor	**1023**	**26**
ExDesc	**1023**	**80**
InstrumentTag	**1023**	**32**
PointSource	**1023**	**1**
Note: If the actual data stored for the point attribute is greater than the maximum field length specified in the table, the returned value for the attribute will be truncated to the maximum length for the given Historian Server version.		

Interfaces Using the PI SDK

Interfaces utilizing the Historian SDK may or may not support disconnected startup. Some interfaces by default create a Historian SDK connection to retrieve data from the Historian Server. The operational need for the Historian SDK is interface specific. For example, the PI SDK might be used to create Historian Points, collect information from components such as the Module Data Base (MDB), and/or retrieve extended point attribute information. Use of the PI SDK by the interface to retrieve the extended attribute information from the host Historian Server is not supported. The disconnected startup feature utilizes the PI API to retrieve extended point information from the Historian Server that is stored in the necessary caching files. Interfaces supporting disconnected startup will explicitly say that disconnected startup is supported in the “Supported Features” table of the interface manual.

As a general rule, disconnected startup will not be supported if the Historian SDK is needed for the interface to start. However, an interface supporting disconnected startup as indicated in the supported features table which uses the PI SDK by default to retrieve extended point attribute information from the Historian Server will require additional configuration to take advantage of the disconnected startup feature. If the host Historian Server is earlier than version 2, the user must define **`/pisdk=0`** in the startup command file to disable and prevent the interface from using the Historian SDK to retrieve extended attribute information from the Historian Server. If the host Historian Server is 2 or later, the interface will always use the Historian API to retrieve extended attribute information and not require the Historian SDK to be disabled.

For Example, the Historian OPC Interface by default is designed to create a PI SDK connection. If the host Historian Server is earlier than version 2, the interface by default will use the Historian SDK to retrieve extended point information from the Historian Server. To use the disconnected startup feature, the user can force the use of the Historian API to retrieve extended point information from the Historian Server instead of using the Historian SDK by defining **`/pisdk=0`** in the startup command file. The extended attribute field lengths will be limited by the Historian Server version used as shown in the table listed in the preceding "Extended Attribute Length" section.

Outputs

The interface can be operational in the disconnected startup configuration while the Historian Server is unavailable. Therefore, outputs will not be supported while disconnected from the Historian Server. Once the interface establishes a valid connection the Historian Server, outputs from the Historian Server to the interface will be supported.

Startup Command File Configuration

Note: The disconnected startup feature can not be used if the interface is using the PI SDK to retrieve extended point attribute information for points from the Historian Server. Either disable the PI SDK by defining **/pisdk=0** in the startup command file or remove any disconnected startup parameters from the interface startup command file. Failure to implement one of the preceding options with the PI SDK configured to retrieve point attribute information will cause the interface to log an error message in the pipc.log file and then shutdown.

Some UniInt interfaces are designed to require the use of the PI SDK to retrieve point information if the Historian API is less than version 1.6 and/or the Historian Server version is less than 2.x. In this case, setting the **/pisdk=0** startup command-line parameter may not disable the PI SDK and prohibit the use of the disconnected startup feature.

There are two interface startup parameters that control the disconnected startup operation: **/CacheMode** and **/CacheSynch**. Each of these parameters is described in the following table.

Parameter	Description
/CacheMode Required Default: `Not Defined`	Required for disconnected startup operation. If defined, the **/CacheMode** startup parameter indicates that the interface will be configured to utilize the disconnected startup feature.
/CacheSynch=# Optional Default: 250 ms	**NOTE:** Care must be taken when modifying this parameter. This value must be less than the smallest scan class period defined with the **/f** parameter. If the value of the **/CacheSynch** parameter is greater than the scan class value, input scans will be missed while the point cache file is being synchronized. The optional **/CacheSynch=#** startup parameter specifies the time slice period in milliseconds (ms) allocated by UniInt for synchronizing the interface point cache file with the Historian Server. By default, the interface will synchronize the point cache if running in the disconnected startup mode. UniInt allocates a maximum of # ms each pass through the control loop synchronizing the interface point cache until the file is completely synchronized. Synchronization of the point cache file can be disabled by setting the value **/CacheSynch=0**. The minimum synchronization period when cache synchronization is enabled is 50ms Whereas, the maximum synchronization period is 3000ms (3s). Period values of 1 to 49 will be changed by the interface to the minimum of 50ms and values greater than 3000 will be set to the maximum interval value of 3000ms. Default: 250 ms Range: {0, 50 – 3000} time in milliseconds Example: **/CacheSynch=50** (use a 50ms interval) **/CacheSynch=3000** (use a 3s interval) **/CacheSynch=0** (do not synchronize the cache)

Sample Interface Startup Files

The following is an example of an interface configured for Disconnected Startup. In this example, the interface name is ifc and the interface executable is ifc.exe. Any additional command-line parameters needed for the interface would be defined on the startup command-line file. The startup command file for the interface would be defined as follows:

```
ifc.exe /PS=E /ID=1 /CacheMode /CacheSynch=100 /host=PISrv:5450 ^
other parameters as required
```

ICU Configuration

The use of the ICU is the recommended and safest method for configuring the Interface for Disconnected Startup. With the exception of the notes described in this section, the Interface shall be configured with the ICU as described in the "Configuring the Interface with the ICU" section of this manual. The rest of this section refers to the latest version of the ICU 1.4.1.x or later which requires SDK 1.3.4.333 or greater to use the ICU to configure disconnected startup. The following figure shows the Disconnected Startup configuration screen of the ICU.

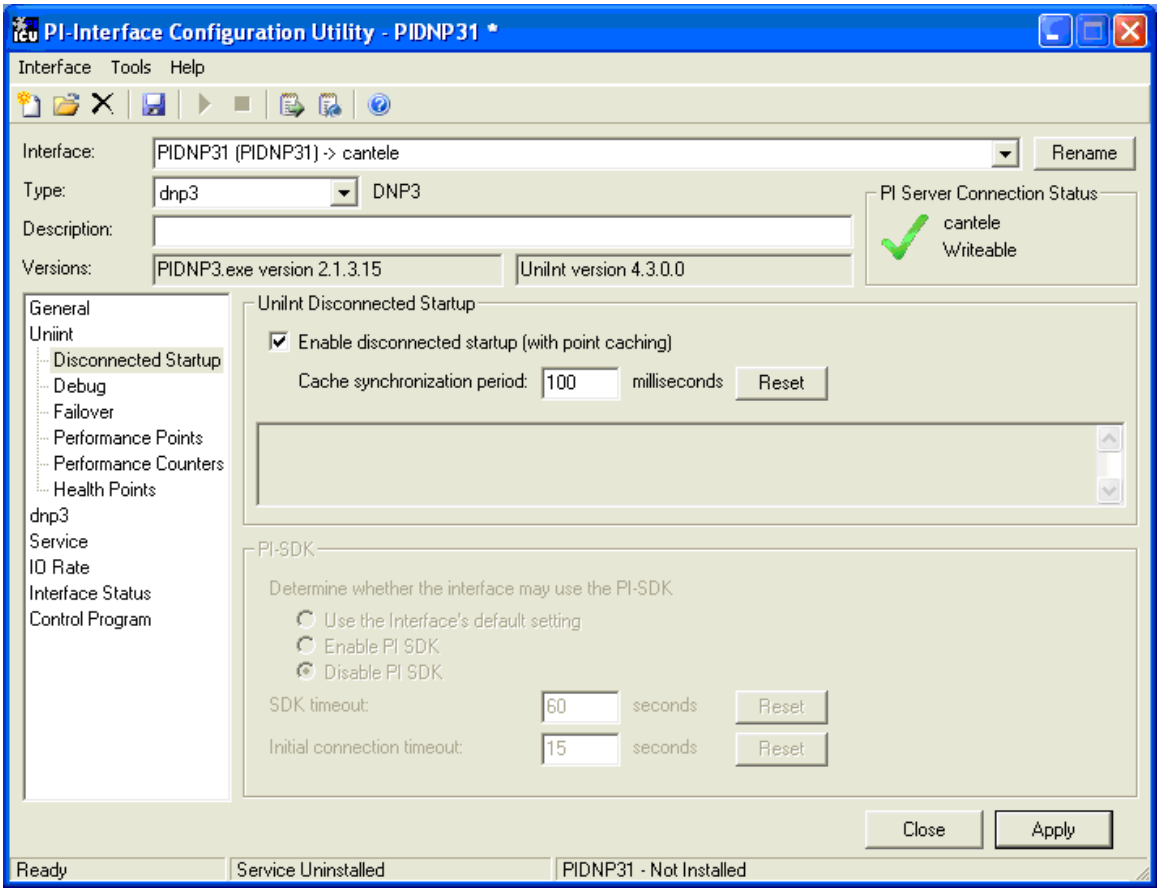


Figure 1: ICU configuration screen showing the UniInt Disconnected Startup parameters. Configuring for Disconnected Startup simply requires the user to select the option and enter a cache synch period if the default of 250ms is not desired. The PI SDK option is disabled and the /pisdk=0 parameter will be defined in an effort to disable the PI SDK.

The synchronization period is limited to a value of 0 or a range of values from 50 to 3000 ms inclusive. Values outside this range will generate an informational message indicating the out or range error. The following figure in an example of the message generated when an invalid value is entered into the Cache synchronization period text field.

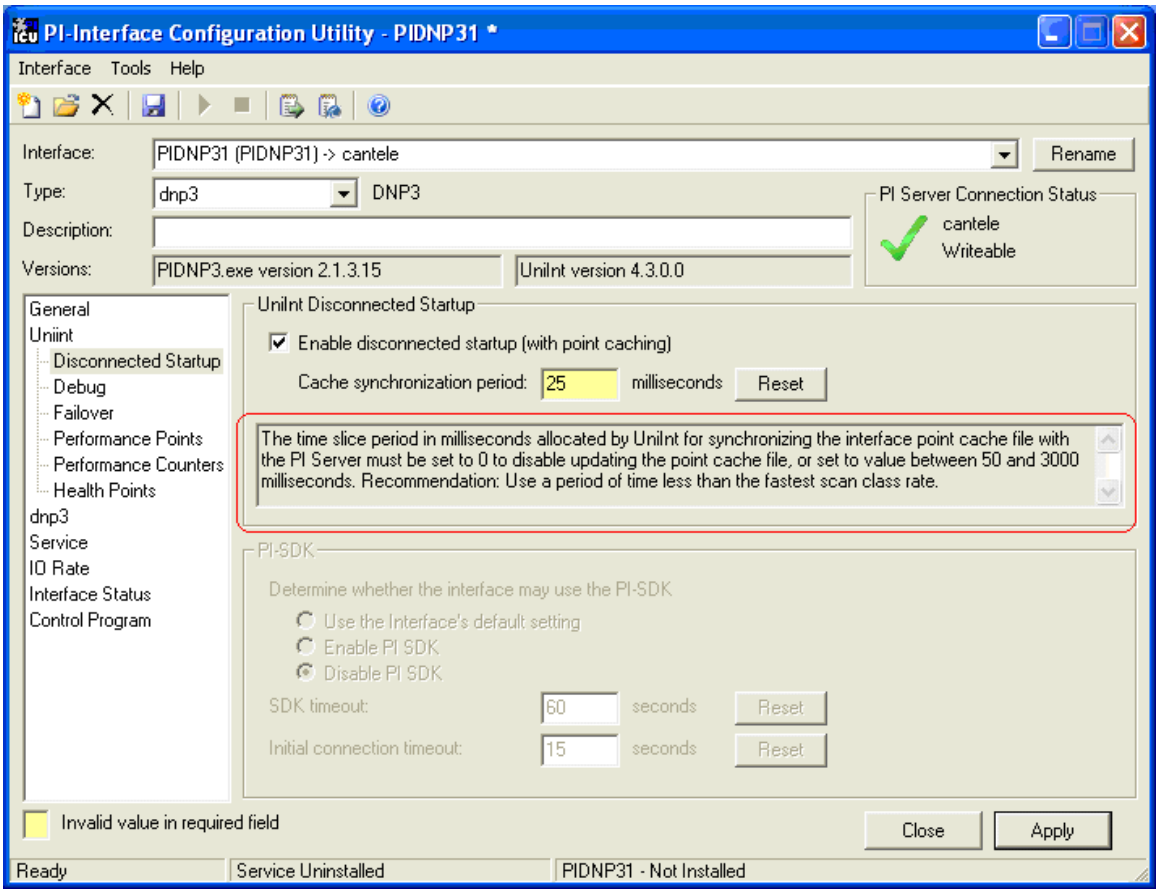


Figure 2: ICU configuration screen displaying a message that the time slice entered for the "Cache synchronization period" text field is out or range.

To run in a disconnected startup configuration, the PI SDK must not be enabled. If the ICU has the "Enable PI SDK" radio button selected, the "UniInt Disconnected Startup" feature will be disabled. The following figure shows how the ICU display will look when the PI SDK is enabled.

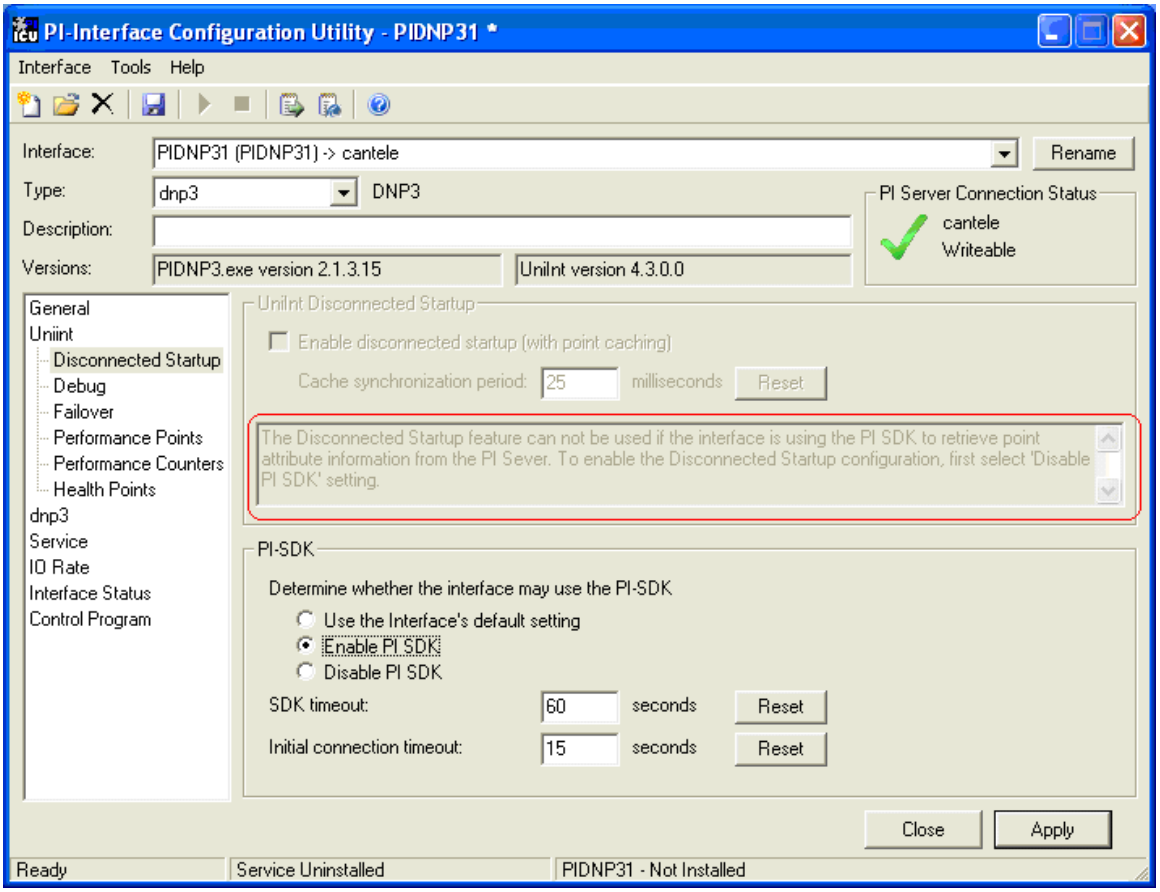


Figure 3: ICU configuration screen with the "Enable PI SDK" radio button selected. In this configuration, the "UniInt Disconnected Startup" feature is disabled and contains a message prompting the user to first disable the PI SDK if the disconnected startup feature is desired.

UniInt version 4.3.0.x or later must be installed on the interface node to support the disconnected startup configuration. The following figure shows how the ICU display will look when the UniInt version is not suitable for the disconnected startup feature.

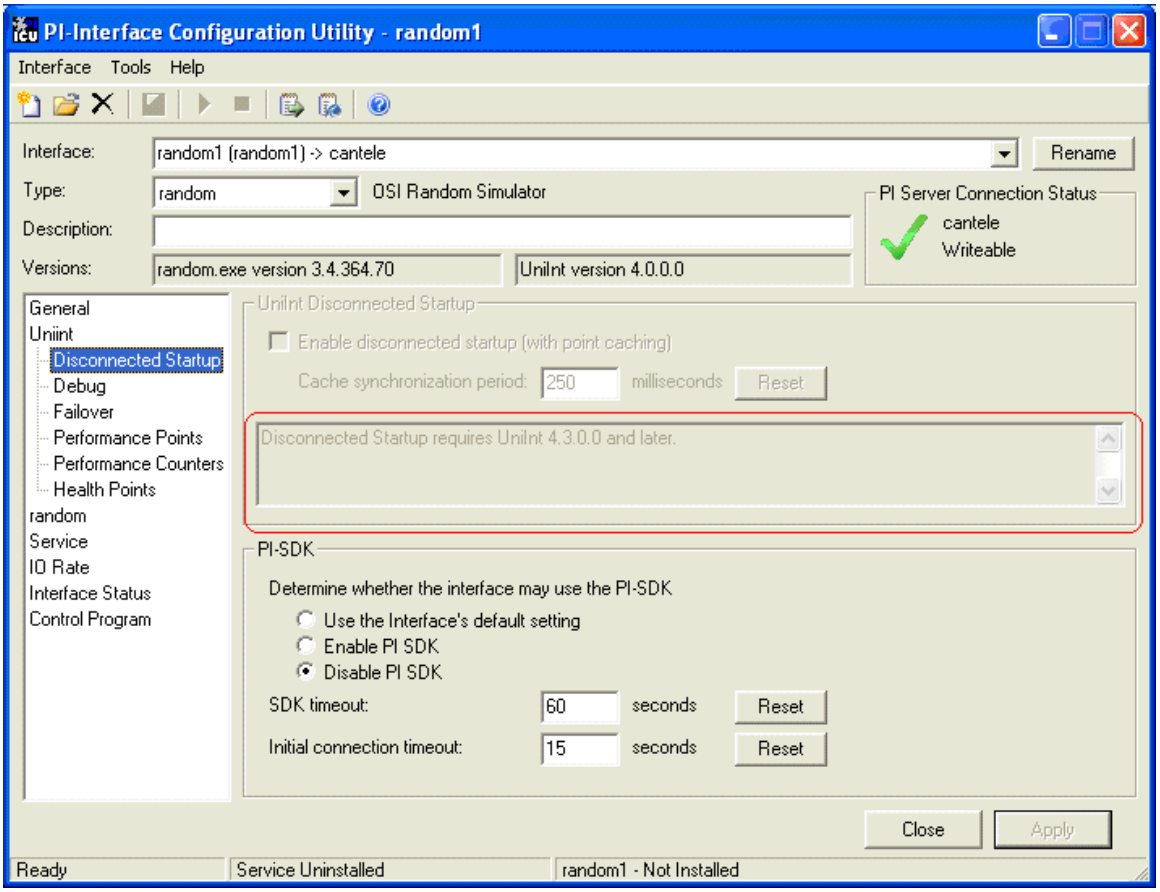
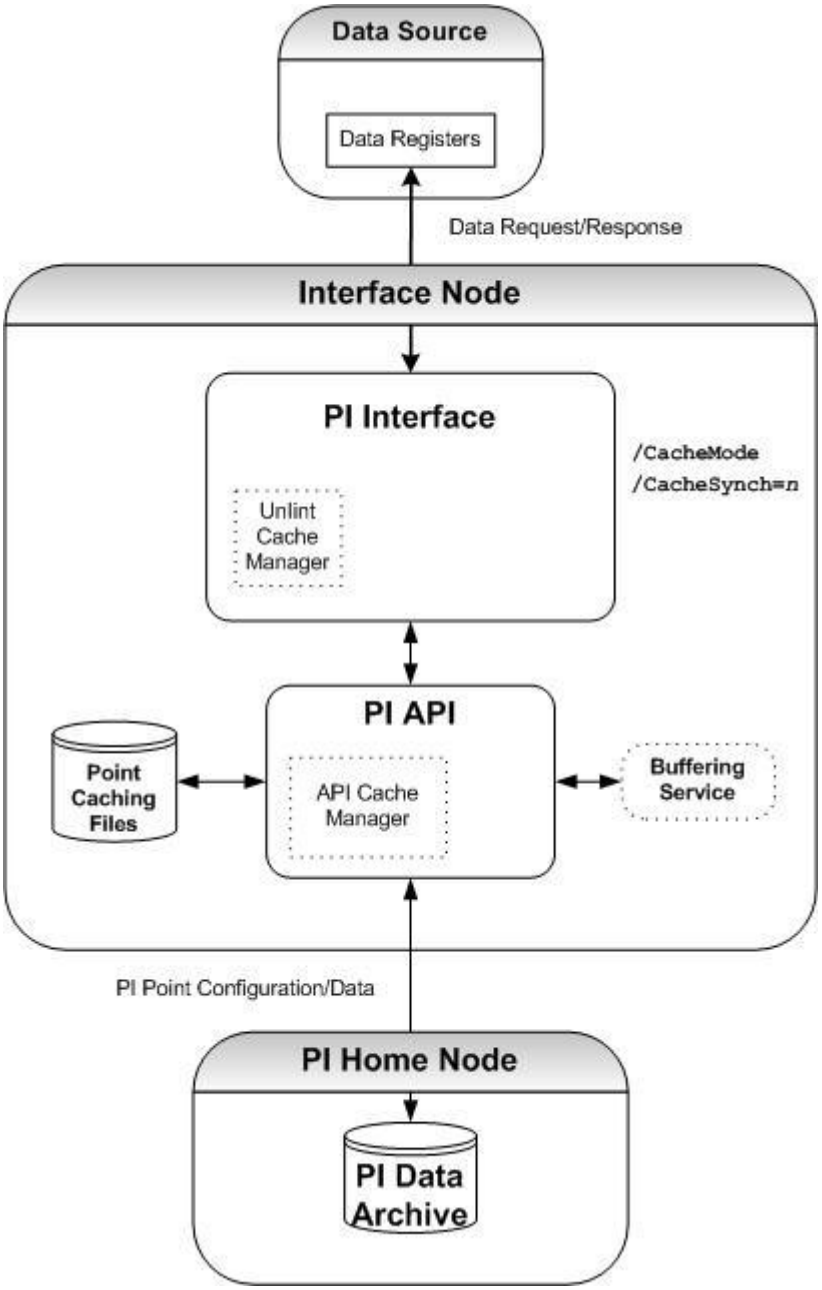


Figure 4: ICU configuration screen with UniInt version 4.0.0.0. The UniInt version must be 4.3.0.x or later in order to support disconnected startup. In this configuration, the "UniInt Disconnected Startup" feature is disabled and the ICU displays a message indicating that disconnected startup requires UniInt 4.3.0.x or later.

Disconnected Startup Architecture

The architecture in the following figure shows the typical components of a UniInt interface utilizing the Disconnected Startup feature. In the figure, the Data Source contains the time-series data of interest to the interface. The Historian Server manages point configuration information, maintains a historical record of time-series data, and provides a means for accessing real-time data. The Interface Node provides a platform for the Historian Interface, PI API, and the Point Caching Files necessary to operate in a Disconnected Startup configuration. The buffering service provides a means to buffer collected data when a connection to the Historian Server in unavailable. Within the Historian Interface there is a UniInt Cache Manager that directly communicates with the API Cache Manager to control and monitor the disconnected startup operation. The API Cache Manager is responsible for maintaining the Point Caching Files necessary to start the interface in a disconnected startup configuration.



Operation

To configure the interface for disconnected startup operation, the required **/CacheMode** parameter must be defined in the interface startup command file. The interface will create two binary caching files on the interface node necessary for disconnected startup. These files require no user configuration. The disconnected startup configuration consists of five distinct modes of operation: creation, validation, construction, utilization, and synchronization. Each of these modes is further described in the following sections.

Creation

CAUTION: The interface will create a point cache file and digital set cache file necessary for the disconnected startup configuration. These files are automatically generated by the interface and are locked while the interface is running. To prevent data loss, these files must not be modified by the user when the interface is not running.

The creation mode ensures the interface has the necessary cache files needed for the disconnected startup configuration. The two files necessary for this configuration are the point cache file and digital set cache file. If one or both of these files is missing, the missing file or files will be created. The files by default are created in the same location as the interface executable. If the path to the executable cannot be established, the Historian API will provide a default location based on information available to the application. In all cases, the filename and location are logged in the pipc.log file. The "Messages" section of this document contains a listing of typical messages sent to the pipc.log file during the creation mode. The file name is constructed using the executable name, host name, point source, and id provided by the startup command file. The point cache file will have the form

filepath/exename_hostname_pointsource_id_ptcache.dat

Likewise, the digital cache file will have the form

filepath/exename_hostname_pointsource_id_dgcache.dat

For example, if the file path to the executable is "`C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC\Interfaces\Test`", with an executable name of "`Test`", host name of "`FreeBird`", point source of "`abc`", and an id of "`1`", the point cache and digital cache files created in the directory indicated by the path to the executable would have the following name:

c:/program files/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat

c:/program files/pipc/interfaces/test/test_freebird_abc_1_dgcache.dat

Part of the creation process includes storing Interface and Historian Server versioning information in the each of the files to be used in the validation process. Therefore, a connection to the Historian Server must be available during the initial creation process for the disconnected startup operation to be configured.

Note: If either of the caching files fails to be created, an error message will be logged to the pipc.log file and the interface will fail to start.

Validation

CAUTION: Users must not modify the caching files created by the interface. User modification of either of the caching files has the potential of invalidating the file and inducing a data loss situation.

If a file has been modified by the user, it should be deleted to allow the interface to create valid files necessary for the Disconnected Startup configuration. Newly created files will require a connection to the Historian Server to start the interface.

An interface starting in the disconnected startup configuration requires the caching files be validated to determine their usability. The validation process ensures the point and digital set cache files are designated for a particular instance of the interface and the files are capable of receiving new data. If the cache files are validated, the process will then provide the interface with the stored Historian Server versioning information necessary to start the interface. Finally, the validation process determines if the cache file has stored all point information needed to start the interface without a connection to the Historian Server. If the validation process determines the cache file is not complete, the interface will wait for a connection to the Historian Server to become available. The Historian Server connection is needed to finish initializing the necessary cache files. The status of the validation process is logged in the pipc.log file.

A file which cannot be validated will be renamed and a new file will be created as described in the "Creation" section. Renamed files will contain an inserted numerical date-time change string in the filename. If either of the point or digital set cache files cannot be found, the necessary file(s) will be created as previously described. The "Messages" section of this document contains a listing of typical messages, which includes the renaming of an invalid file, sent to the pipc.log file during the validation mode.

Construction

Note: If one or both of the cache files becomes invalid at any time during the construction process, the interface will disable the disconnected startup feature and continue to operate in the normal operation profile. During normal operation, a connection to the Historian Server is required for the interface to start.

This construction mode of operation is where the cache files are populated with the necessary digital state and point attribute data.

Construction of the digital cache file is determined as follows. If the interface requires caching of the system digital state set, then all the system digital states will be retrieved from the Historian Server and stored in the digital cache file. If the interface receives a digital point from the Historian Server or queries the server for a digital point, all the digital states for the digital set associated with the Historian point will be cached. Multiple points with the same digital state set will only require the complete digital set to be cached once.

The point cache file is constructed in a similar fashion to the digital cache file. During interface startup, the interface requests and receives all points matching the required point source parameter listed in the startup command file. Upon receipt of the requested points from the Historian Server, the point record along with the associated extended attributes are requested from the Historian Server and immediately stored in the point cache file. The interface may request additional points such as event counters or trigger points. These points are cached as well. As a rule, any point requested by the interface will result in the point record and associated extended attributes being cached.

The initial construction of the cache files will require additional startup time for the interface. When constructing the digital cache file, network latency may add to the startup time if the entire system digital set table is cached because an additional 1,024 network calls (one for each possible system digital state) will need to be executed because the interface is only capable of retrieving a single digital state from the Historian API for a given call. Historian Points with large digital sets will also require additional caching time. The caching of the point record and extended attributes will add additional startup time to the interface as well.

However, restarting the interface in a disconnected startup configuration with a fully constructed point cache file will eliminate all network calls and latency associated with communicating with the Historian Server. This will dramatically reduce the interface startup time for subsequent interface restarts and allow the interface to collect data from the Data Source much quicker than without running in a disconnected startup configuration.

The "Messages" section of this document contains a listing of typical messages sent to the pipc.log file during the construction mode.

Utilization

Note: If one or both of the cache files becomes invalid at any time during run-time, the interface will disable the caching feature of the disconnected startup configuration and continue to operate in the normal non-caching operating profile. During normal operation, a connection to the Historian Server is required to retrieve point update information.

The utilization mode is the run-time use of the caching files that have been validated and fully constructed. In this mode, all requests for digital state or point attribute data will be routed to and managed by the API Cache Manager. Requested data will be retrieved from the appropriate cache file and returned to the interface. While the cache files are in a fully configured and valid state, no network calls will be made to the Historian Server to retrieve the requested information.

The utilization process using fully configured cache files allow the following run-time operations without requiring a connection to the Historian Server.

- Validate cache files to be used for interface instance.
- Retrieve Historian Server version information.

- Retrieve event counter point configuration.
- Retrieve system digital state set if requested.
- Retrieve point configuration for points matching interface point source value. This includes retrieving the digital state set for digital points.
- Retrieve additional points required by interface. (Trigger, Source, etc.)

The interface will request point update information from the Historian Server at regular intervals. The API Cache Manager will process updates received and modify the point cache file if necessary to keep the cache files up to date. The update(s) will then be returned to the interface for proper handling.

Synchronization

During interface disconnected startup with fully configured cache files, only the cache files will be used to configure the interface with the necessary data to begin data collection. If the interface has been disconnected from the Historian Server, the point configuration data stored in the cache files may no longer match what is currently on the Historian Server. Point attribute changes may have been made or points matching the interface point source may have been added or deleted. Therefore, a synchronization mechanism is required to make sure the Historian point configuration data running on the interface is up to date. The synchronization mode keeps the caching files up to date and passes any changes in point configuration to the interface for processing.

The synchronization process can only begin after the interface has been successfully started using the disconnected startup configuration, entered its control loop, and established a connection to the Historian Server. Once these conditions are satisfied, the UniInt Cache Manager will request the API Cache Manager to retrieve a list a Historian Points matching the interface point source from the Historian Server. If a point received from the Historian Server does not exist in the point cache, it will be added to the cache and then a point create update will be sent to the interface. If a point has been modified as indicated by the "`changedate`" attribute, the cache will be updated with the correct attribute information and a point edit update will be sent to the interface. A comparison of the points found on the server is made to what currently exists in the caching files. If a point is found in the cache file that no longer exists on the Historian Server, a point delete update will be sent to the interface.

The optional **`/CacheSynch`** startup parameter, described in the Startup Command File Configuration section, specifies whether cache synchronization is enabled and the time period allocated to synchronization. By default, synchronization of the cache files is enabled and the time period allocated for synchronizing the cache is set to a default of 250 milliseconds (ms) per synchronization attempt. This means one iteration through the interface control loop can use a maximum of 250ms to synchronize as many points as it can. The synchronization process will continue in this manner until all the points have been synchronized. Once the cache files have been synchronized, cache synching will be disabled unless there is an interruption with the connection with the Historian Server. If the Historian

Server becomes disconnected for any reason, point synchronization will be restarted upon reconnection to the Historian Server.

The "Messages" section of this document contains a listing of typical messages sent to the pipc.log file during the synchronization mode

Remote Cache Builds

Users have the option of building the cache files necessary for disconnected startup on a remote machine. This feature is useful if the destination machine has limited network connectivity with the Historian Server or if identical interface configuration will be deployed as in a failover configuration using the same host Historian Server. For example, if the interface node is running on a remote off-shore oil platform using a network connection with limited throughput, the cache files may be built on an interface node or even the Historian Server located at the corporate headquarters. The files may then be transferred to the remote site as desired. The cache files built on the remote machine must be validated by the destination interface node. To validate remotely created cache files, the interface startup command file (.bat file) used in creating the remote cache files must contain identical entries for the parameters necessary for the disconnected startup configuration in the startup command file to be used on the destination interface node. The following is a list of requirements necessary to successfully build caching files on a remote machine.

- The interface must be built against UniInt version 4.3.0.x or later.
- The interface node or Historian Server must have PI API version 1.6.1.x or later installed.
- The PI SDK must not be utilized to retrieve the extended point attributes from the Historian Server.
- The following entries on the startup command file must be identical on both the remote and destination nodes. The entries are not case sensitive.
    - The interface executable name.
    - The host Historian Server name. Use the actual host Historian Server name or IP Address. Do not use **`/host=localhost`**.
    - The **`/ps`** parameter.
    - The **`/id`** parameter.
    - The **`/cachemode`** parameter must be defined.

Messages

The following are examples of typical informational and error messages that can be found in the pipc.log file.

Informational

```
21-Nov-06 16:57:51
WriteSeconds.exe>Historian-API> Attempting to initialize and validate cache file:
       c:/pipc/interfaces/test/test_freebird_abc_1_dgcache.dat.
```

Meaning: The API Cache Manager will look for the digital cache file in the given path and attempt to initialize and validate the file. If the file is not found, an attempt will be made to create the file with the file path and filename shown in the message.

```
21-Nov-06 16:57:51
WriteSeconds.exe>Historian-API> Attempting to initialize and validate cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat.
```

Meaning: The API Cache Manager will look for the point cache file in the given path and attempt to initialize and validate the file. If the file is not found, an attempt will be made to create the file with the file path and filename shown in the message.

```
21-Nov-06 16:54:41
WriteSeconds.exe>Historian-API> Successfully validated cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_dgcache.dat
```

Meaning: The API Cache Manager was able to validate the existing digital cache file. A valid cache file means the digital cache file is designated for this particular instance of the interface, contains the host Historian version information, and that the files are capable of receiving new data. The file may or may not have been completely constructed with the necessary point information.

```
21-Nov-06 16:54:41
WriteSeconds.exe>Historian-API> Successfully validated cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
```

Meaning: The API Cache Manager was able to validate the existing point cache file. A valid cache file means the point cache file is designated for this particular instance of the interface, contains the host Historian version information, and that the files are capable of receiving new data. The file may or may not have been completely constructed with the necessary point information.

```
21-Nov-06 16:57:51
WriteSeconds.exe>Historian-API> Successfully created cache file:
       c:/pipc/interfaces/test/test_freebird_abc_1_dgcache.dat.
```

Meaning: The API Cache Manager was able to create a file with the file path and filename shown in the message. This message also means that the interface successfully connected to the Historian Server to retrieve versioning information needed to create the cache file. The file has not been constructed with digital state information and will continue to require a connection to the Historian Server.

```
21-Nov-06 16:57:51
WriteSeconds.exe>Historian-API> Successfully created cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat.
```

Meaning: The API Cache Manager was able to create a file with the file path and filename shown in the message. This message also means that the interface successfully connected to the Historian Server to retrieve versioning information needed to create the cache file. The file has not been constructed with point information and will continue to require a connection to the Historian Server.

```
21-Nov-06 16:54:41
WriteSeconds.exe>Historian-API> Point information will be loaded from cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
```

Meaning: The API Cache Manager has determined the point and digital cache files have been completely constructed and will be used to load point information. No connection to the Historian Server is necessary for the interface to start.

```
21-Nov-06 16:54:41
Historian-WriteSeconds 1> Point cache being used to retrieve point information.
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
   Cache synchronization enabled with interval set to 250 m/s.
```

Meaning: The UniInt Cache Manager has determined the point and digital cache files have been completely constructed and will be used to load point information. No connection to the Historian Server is necessary for the interface to start.

This message also states the cache will be synchronized with a 250ms period.

```
21-Nov-06 17:00:04
WriteSeconds.exe>Historian-API> Renamed cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_dgcache.dat
    to
   c:/pipc/interfaces/test/test_freebird_abc_1_dgcache_20061121220004.dat.
```

Meaning: The API Cache Manager has determined the current digital cache file is invalid and unusable. The file has been renamed as indicated in the message. Expect to see a message indicating a new digital cache file has been created. A connection to the Historian Server will need to be available for the interface to start.

```
21-Nov-06 17:00:04
WriteSeconds.exe>Historian-API> Renamed cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
    to
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache_20061121220004.dat.
```

Meaning: The API Cache Manager has determined the current point cache file is invalid and unusable. The file has been renamed as indicated in the message. A message indicating a new point cache file has been created will be printed to the pipc.log file. A connection to the Historian Server is required for the interface to start.

```
21-Nov-06 16:57:51
WriteSeconds.exe>Historian-API> Cache valid for reading and writing, but not complete.
   Point information will be retrieved from the Historian Server and saved in cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat.
```

Meaning: The API Cache Manager was able to create and validate the point

cache file, but the file has not been constructed with point record and attributes information. This message also indicates the digital cache will need to be constructed with digital state information. A connection to the server is required for the interface to start so the file can be constructed.

```
21-Nov-06 16:57:51
Historian-WriteSeconds 1> Point cache being constructed to store point information.
   Cache file: c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat.
   Cache synchronization enabled with period set to 250 ms.
```

Meaning: The UniInt Cache Manager has detected a point caching file that has not been constructed with point record and attributes information. This message also indicates the digital cache will need to be constructed with digital state information. A connection to the server is required for the interface to start so the file can be constructed.

The message also indicates that cache synchronization is enabled with a time period set to 250ms.

```
21-Nov-06 16:57:51
Historian-WriteSeconds 1> Begin synching cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
```

Meaning: The UniInt Cache Manager has loaded point information from the cache, entered the control loop, connected to the Historian Server and begun cache synchronization. The digital cache has also begun synchronization.

```
21-Nov-06 16:57:51
Historian-WriteSeconds 1> Completed synching cache file:
   c:/pipc/interfaces/test/test_freebird_abc_1_ptcache.dat
```

Meaning: The UniInt Cache Manager has completed point and digital cache synchronization. Updates will be sent to the interface.

Configuration Errors

```
22-Nov-06 16:21:19
Historian-WriteSeconds 1> Error = -1: Point caching configuration error.
   The PI SDK is being used to retrieve point attribute information.
   Unable to utilize point caching while in this mode.
   Disable cachemode by removing "/cachemode" parameter from the
   startup command file and restart the interface.
   Stopping Interface
```

Meaning: The UniInt Cache Manager has determined the PI SDK is being used to retrieve point attribute information while attempting to start the interface in a disconnected startup mode. Therefore, caching will be disabled and the interface will be shutdown.

Resolution: Option 1: Remove the `/cachemode` startup command parameter and restart the interface.

Option 2: Try using the `/pisdk=0` startup command parameter to disable the interface. Restart the interface.

Note: not all interfaces can disable the PI SDK using the `/pisdk=0` option.

```
29-Nov-06 15:45:08
Historian-WriteSeconds 1> Disconnected Startup is not available with PI API prior
   to Version: 1.6, Build 1.5
   To utilize the disconnected startup configuration, upgrade to
   PI API Version: 1.6, Build 1.5 or later and restart the interface.
   Stopping Interface.
```

Meaning: The UniInt Cache Manager has found a PI API version currently installed on the interface node that does not support disconnected startup. The interface has been configured with the `/CacheMode` startup parameter and will be stopped.

Resolution: Option 1: Upgrade the PI API on the interface node to version 1.6.1.x or later. Restart the interface.

Option 2: Remove the `/CacheMode` startup command parameter. Restart the interface.

```
29-Nov-06 15:56:19
Historian-WriteSeconds 1> Error: 0, Setting cache function pointer failed.
   Unable to load picm_closecache from API dll
   Point caching unavailable for this interface.
   Correct the error or remove the /cachemode startup command parameter.
   Stopping Interface.
```

Meaning: The UniInt Cache Manager has failed to load a required function from the PI API. The interface has been configured with the `/CacheMode` startup parameter and will be stopped.

Resolution: Option 1: Upgrade the PI API on the interface node to version 1.6.1.x or later. Restart the interface.

Option 2: Remove the `/CacheMode` startup command parameter. Restart the interface.

```
29-Nov-06 15:56:19
Historian-WriteSeconds 1> Error 0: The /uiinfops must be defined with a single
   character
   pointsource value on the startup command-line when using /cachemode
   against a Historian Server version ealier than 2.
   Intializing uniint library failed
   Stopping Interface
```

Meaning: The `/uiinfops` startup command-line parameter is either missing or has a multi-character point source designated for the interface information tag. This error occurs when running in a disconnected startup configuration and connecting to a host Historian Server version earlier than 2. The interface will be shutdown.

Resolution: Option 1: Define the `/uiinfops` startup command-line parameter with a single character point source. Restart the interface.

Option 2: Remove the `/CacheMode` startup command parameter to run in a non-disconnected startup configuration. Restart the interface.

Troubleshooting

Interface Operation

Problem:

On restart, the interface is not loading all Historian Points with a fully configured cache file.

Solution:

1. Make sure the cache files have been previously configured during the initial interface start using the disconnected startup configuration. This can be verified by reviewing Historian point value history on the Historian Server using the appropriate Historian client, such as DataLink and by reviewing the pipc.log file that contains the initial start of the interface configured for disconnected startup. The Historian Points for the interface should have been receiving data during the initial cache build process. The user should see messages containing cache file information as described in the "Messages" section of this document. If cache synching is enabled, there will be a message stating the cache file synchronization is complete. There should also be a message containing the number of points matching the point source for the interface. If no modifications to the Historian Server database for Historian Points matching the interface point source have been made since the cache files have been built, the number of points for the interface starting from the cache file should be identical.

2. Restart the interface.

3. Confirm the number of points matching the point source is identical to the number of points matching the point source on the Historian Server.

4. If the number of points loaded by the interface does not match the number of points on the Historian Server, attempt additional restart. If a restart is not successful, rename or delete the point caching files to allow the interface to create new files on the next restart.

5. Contact Rockwell Automation technical support if none of the above procedures results in a successful disconnected startup configuration.

Deleting Cache Files

Rockwell Automation recommends renaming or moving the cache files to a backup directory. Delete cache files only if absolutely necessary.

To delete the cache files created by the interface. The following procedure is recommended.

1. Stop the interface.

2. Verify the location and names of the cache files belonging to the interface. There are two files: a point cache file and digital cache file. The location and exact name of these files will be listed in the pipc.log file for the interface instance.

3. Delete the cache files.

4. Restart the interface to generate new cache files.

UniInt Failover Scheme

Introduction

The UniInt failover scheme provides for a hot failover, no data loss solution given a single point of failover. The scheme requires the data source be able to communicate and service data to two interfaces simultaneously. Additionally, the failover configuration requires the interface supports outputs.

UniInt interface level failover may be configured to send data to a H.A. Historian Server collective. The H.A. Historian Server basically provides redundant Historian Servers to allow for the un-interuppted collection and presentation of Historian Time series data while a single Historian Server can be taken down for maintenance or in the case of a hardware error that causes a single Historian Server to become unavailable. The H.A. Historian Server collective is described in the *Historian Server Reference Guide*.

When configured for UniInt failover, the interface routes all data that needs to be sent to Historian through a state machine. The state machine determines whether to queue the data or send the data to Historian depending on current state of the interface. When the interface is in the primary state, any data sent through the interface will be routed to Historian. When in the backup state, any data sent through the interface will be queued for a short period. The data being queued by the backup interface ensures a no-data loss failover under normal circumstances. The same algorithm is used for output data.

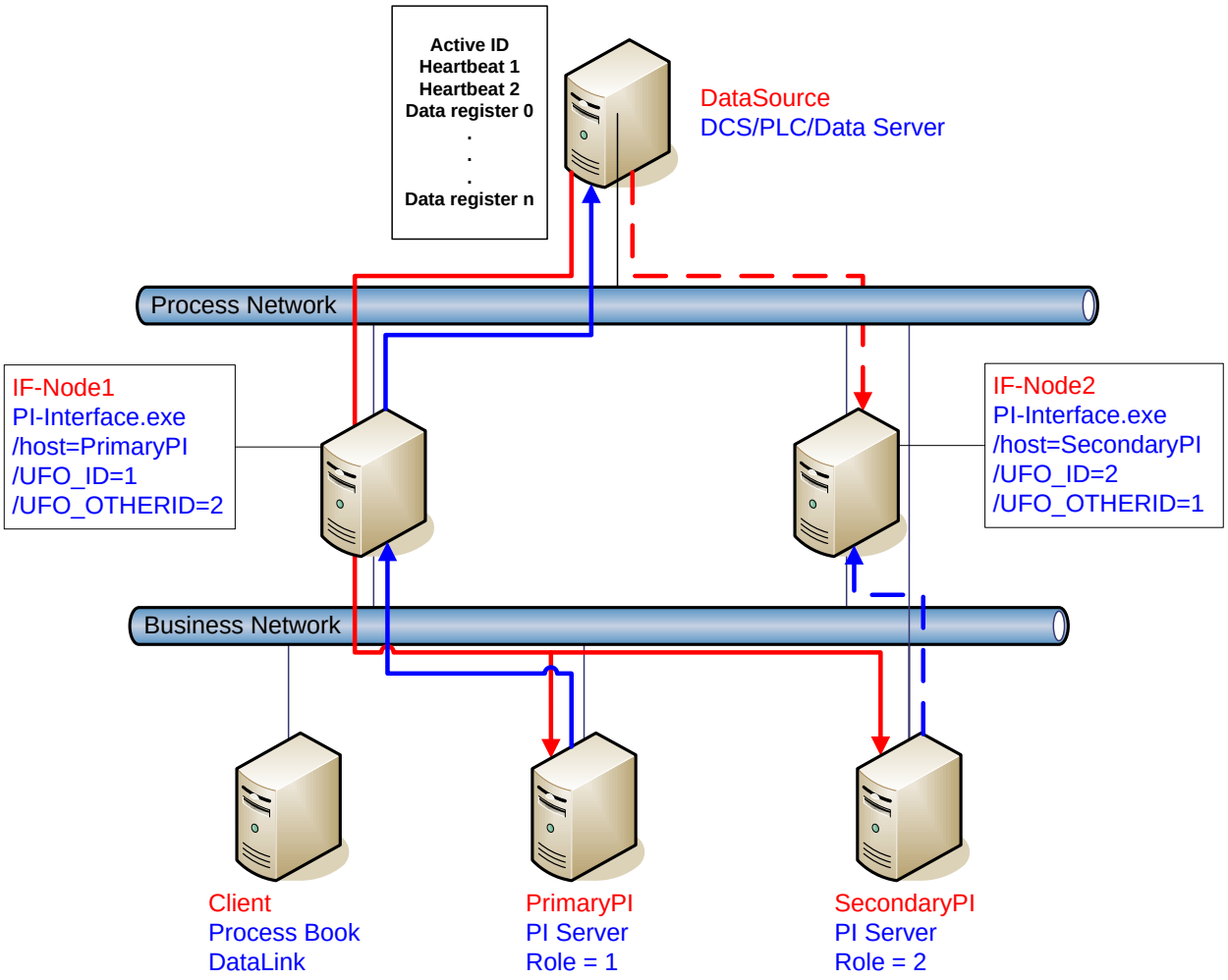


Figure 1: Failover Architecture

The failover architecture is shown in Figure 1. The figure above shows a typical network setup. This by no means includes the myriad of configurations that are supported, it is meant to be used for an example. The figure is repeated and explained in greater detail after the discussion of the required start-up parameters, data source points and Historian tags.

Configuring UniInt Failover

Start-Up Parameters

The following table lists the start-up parameters used by UniInt Failover. All of the parameters are required expect the /UFO_Interval startup parameter. See the table below for further explanation.

Parameter	Value	Description
Failover ID **Computer:** IF-Node1	/UFO_ID=1	The value is not required to be 1. The value must be a positive, non-zero integer and different from the failover ID of IF-Node2.
Failover ID **Computer:** IF-Node2	/UFO_ID=2	The value is not required to be 2. The value must be a positive, non-zero integer and different from the failover ID of IF-Node1.
Other Failover ID **Computer:** IF-Node1	/UFO_OtherID=2	The value is not required to be 2. The value must be a positive, non-zero integer and equal to the Failover ID configured for the interface on IF-Node2.
Other Failover ID **Computer:** IF-Node2	/UFO_OtherID=1	The value is not required to be 1. The value must be a positive, non-zero integer and equal to the Failover ID configured for the interface on IF-Node1.
Failover Update Interval **Computer:** IF-Node1 and IF-Node2	/UFO_Interval=n **Units for n:** milliseconds **Default:** 1000	The Failover Update Interval specifies the rate at which UniInt updates the Failover Heartbeat tags as well as how often UniInt check on the status of the other copy of the interface. The /UFO_Interval is only used if the Failover Control Input Historian tags are collected on an unsolicited basis. If the Input Historian tags are scanned, the Failover Update Interval is determined by the scan class the tags are associated with. The value of n specifies the Update Interval in milliseconds and must be the same on both interface computers.
Host Historian Server for Exceptions and Historian tag updates **Computer:** IF-Node1	/Host=PrimaryPI	The value of the /host startup parameter depends on the Historian Server configuration. If the Historian Server is not part of a collective, the value of /host must be identical on both interface computers. If the redundant interfaces are being configured

Parameter	Value	Description
Host Historian Server for Exceptions and Historian tag updates **Computer:** IF-Node2	/Host=SecondaryPI	to send data to a Historian Server collective, the value of the /host parameters should equal different members of the collective. This configuration will ensure output continue to be sent to the Data Source if one of the Historian Servers becomes unavailable for any reason. If the redundant interfaces are being configured to send data to a Historian Server collective, the value of the /host parameters should equal different members of the collective. This configuration will ensure outputs continue to be sent to the Data Source if one of the Historian Servers becomes unavailable for any reason.

Data Source Points

The following table lists the points that are required on the Data Source with the values that will be written to the points and comments describing the points.

Point	Value	Description
Active ID Point	Updated by the redundant Interfaces Can be changed manually to initiate a manual failover Values range between 0 and the highest of the Interface Failover IDs (typically 2)	The Active ID Point on the datasource is used by the interfaces to determine which interface is currently sending data to Historian. This point on the Data Source must be initialized so the interfaces's first read of the value is not an error. The Active ID Point may be used to initiate a manual failover. If the current value of the Active ID point is 1, the value can be manually changed to a 2 which will cause the interface of IF-Node1 to transition to the backup role and the interface of IF-Node1 to transition to the primary role.
Heartbeat 1 Point	Updated by the Interface on IF-Node1 Values range between 0 and 31	The Heartbeat 1 Point on the Data Source is updated periodically by the interface on IF-Node1. The interface on IF-Node2 monitors this value to determine if the interface of IF-Node1 has become unresponsive.
Heartbeat 2 Point	Updated by the Interface on IF-Node2 Values range between 0 and 31	The Heartbeat 2 Point on the Data Source is updated periodically by the interface on IF-Node2. The interface on IF-Node1 monitors this value to determine if the interface of IF-Node2 has become unresponsive.

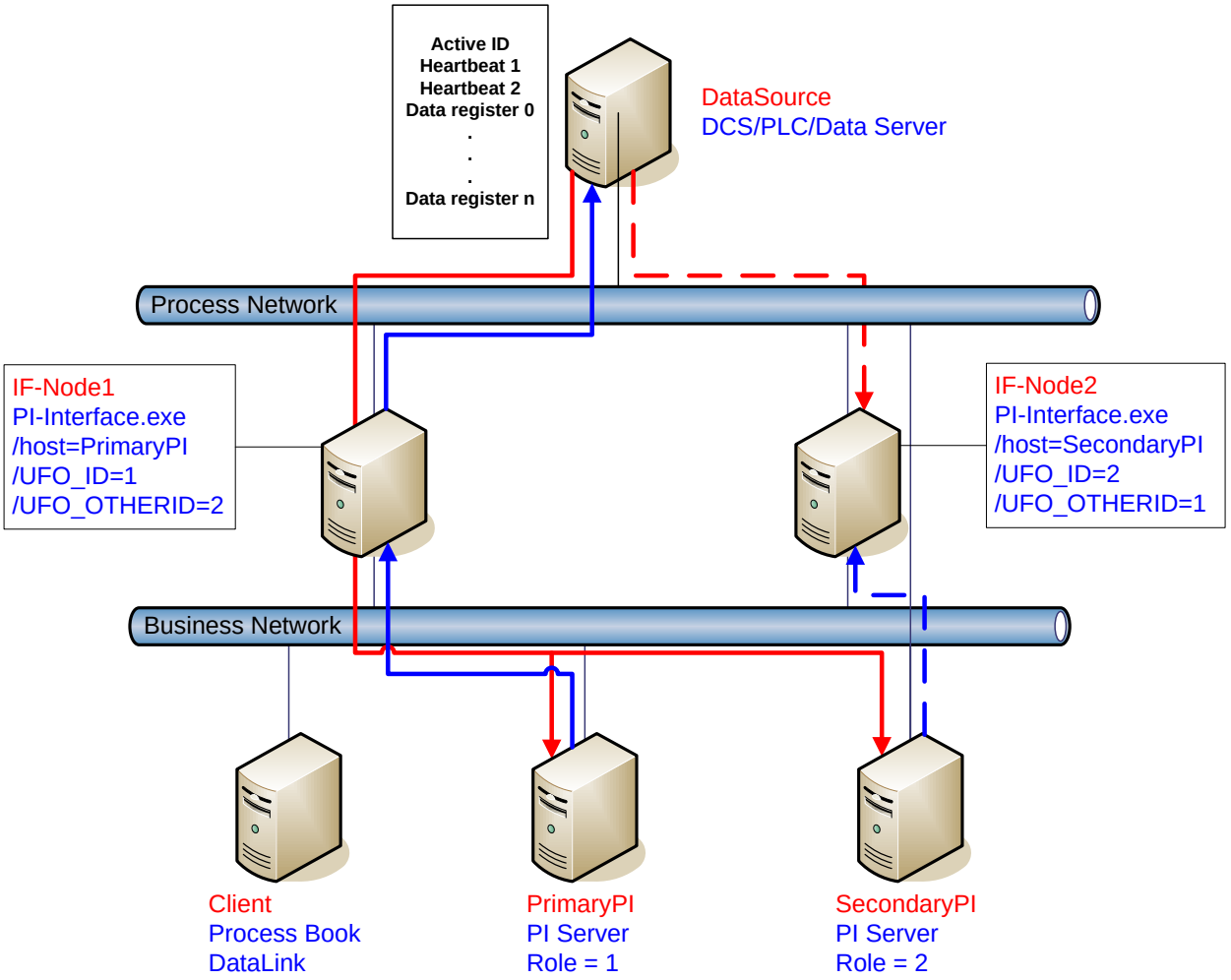
Historian Tags

The following table lists the required UniInt Failover Control Historian tags, the values they will receive and comments.

Historian Tag	Value	Description
Active ID Input Tag Exdesc: [UFO_ACTIVEID]	Updated by the redundant Interfaces Values range between 0 and the highest of the Interface Failover IDs (typically 2)	The Active ID Input Tag must be configured as a valid input Historian tag for the interface and it must be configured to read the Active ID point on the data source. Consult the Interface User's manual for a description of configuring input tags. The exdesc must start with the case sensitive string: [UFO_ACTIVEID]
Active ID Output Tag Exdesc: [UFO_ACTIVEID]	Updated by the redundant Interfaces Values range between 0 and the highest of the Interface Failover IDs (typically 2)	The Active ID Output Tag must be configured as a valid output Historian tag for the interface and it must be configured to write to the Active ID point on the data source. Consult the Interface User's manual for a description of configuring output tags. The exdesc must start with the case sensitive string: [UFO_ACTIVEID]
Heartbeat 1 Input Tag Exdesc: [UFO_HEARTBEAT:1]	Updated by the Interface on IF-Node1 Values range between 0 and 31	The Heartbeat 1 Input Tag must be configured as a valid input Historian tag for the interface and it must be configured to read the Heartbeat 1 Point on the Data Source. Consult the Interface User's manual for a description of configuring input tags. The exdesc must start with the case sensitive string: [UFO_HEARTBEAT:1] The value following the colon (:) must be the Failover ID for the interface running on IF-Node1. In this example, the Failover ID is 1.
Heartbeat 1 Output Tag Exdesc: [UFO_HEARTBEAT:1]	Updated by the Interface on IF-Node1 Values range between 0 and 31	The Heartbeat 1 Output Tag must be configured as a valid output Historian tag for the interface and it must be configured to write to the Heartbeat 1 Point on the Data Source. Consult the Interface User's manual for a description of configuring output tags. The exdesc must start with the case sensitive string: [UFO_HEARTBEAT:1] The value following the colon (:) must be the Failover ID for the interface running on IF-Node1. In this example, the Failover ID is 1.

Historian Tag	Value	Description
Heartbeat 2 Input Tag Exdesc: [UFO_HEARTBEAT:2]	Updated by the Interface on IF-Node2 Values range between 0 and 31	The Heartbeat 2 Input Tag must be configured as a valid input Historian tag for the interface and it must be configured to read the Heartbeat 2 Point on the Data Source. Consult the Interface User's manual for a description of configuring input tags. The exdesc must start with the case sensitive string: [UFO_HEARTBEAT:2] The value following the colon (:) must be the Failover ID for the interface running on IF-Node2. In this example, the Failover ID is 2.
Heartbeat 2 Output Tag Exdesc: [UFO_HEARTBEAT:2]	Updated by the Interface on IF-Node2 Values range between 0 and 31	The Heartbeat 2 Output Tag must be configured as a valid output Historian tag for the interface and it must be configured to write to the Heartbeat 2 Point on the Data Source. Consult the Interface User's manual for a description of configuring output tags. The exdesc must start with the case sensitive string: [UFO_HEARTBEAT:2] The value following the colon (:) must be the Failover ID for the interface running on IF-Node2. In this example, the Failover ID is 2.
State 1 Tag Exdesc: [UFO_STATE:1]	Normally updated by the Interface on IF-Node1. May be updated by the interface on IF-Node2 if IF-Node1 is unable to update the tag Values range between 0 and 5, see description.	The failover state tags are optional and do not require a point on the Data Source. The value of the state tag can be written to the Data Source by configuring a normal interface output tag and setting the SourceTag attribute to the failover state tag. The failover state tags were designed to be digital tag assigned to a digital state set with the value values. 0 = Off: The interface has been shut down. 1 = Backup No Data Source: The interface is running but is unable to communicate to the data source. 2 = Backup No Historian Connection: The interface is running and connected to the data source but has lost its communication to the Historian Server. 3 = Backup: The interface is running and collecting data normally and is ready to take over as primary if the other copy shuts down or experiences problems. 4 = Transition: The interface will be in this state for only a short period of time. The transition period is designed to prevent thrashing where both copies attempt to assume the role of the primary interface. 5 = Primary: The interface is running, collecting data and sending the data to Historian.

Historian Tag	Value	Description
State 2 Tag Exdesc: [UFO_STATE:2]	Normally updated by the Interface on IF-Node2. May be updated by the interface on IF-Node1 if IF-Node2 is unable to update the tag Values range between 0 and 5, see description of State 1 tag.	The failover state tags are optional and do not require a point on the Data Source. The value of the state tag can be written to the Data Source by configuring a normal interface output tag and setting the SourceTag attribute to the failover state tag.



The redundant solution requires two separate interface nodes communicating with the data source. The failover scheme requires the creation of six tags on the Historian Server and three control points on the data source that are specifically used for controlling failover operation. The Historian tags are necessary to initialize the interface with configuration information for reading and writing to the control points on the data source. Once the interface is configured and running, the ability to read or write to the Historian tags is not required for the proper operation of failover. This leads to a solution that does not require a connection to the Historian Server after initial startup in order for failover to operate since

only the control points on the data source are monitored. However, values are echoed to the Historian Server for the user to monitor. Manual failover can be accomplished by manually changing the active ID point on the data source to the appropriate failover ID.

The figure above shows a typical network setup in the normal or steady state. This by no means includes the myriad of configurations that are supported, it is meant to be used for an example. If the hardware configuration differs from the figure, the settings for the two redundant interfaces will remain the same with the exception of the **/host** startup parameter. If the interfaces are communicating to a stand-alone Historian Server, the **/host** parameter for both interfaces must be the same.

In order to ensure outputs to the datasource continue when a Historian Server in the collective becomes unavailable, the interface running on IF-Node1 will need the **/host** parameter set to a Historian Server that is part of the collective, and the interface running on IF-Node2 will need the **/host** parameter set to a **different** Historian Server that is part of the collective. The continued operation of outputs in the face of a Historian Server becoming unavailable assumes the source data for output data (data that is read from Historian and written to the data source) comes into Historian from a process that sends values to all of the Historian Servers in the collective via n-way buffering.

The solid red line in the figure shows input data flow when the interface on IF-Node1 is in the primary state. The data is read from the data source by the interface and sent to buffering. Buffering sends the input data to all of the Historian Servers in the collective via n-way buffering.

The solid blue line shows output data flow. Since the interface on IF-Node1 is configured with **/host=PrimaryPI**, the interface signs up for exceptions with the Historian Server on PrimaryPI. Exceptions are received by the interface and sent to the data source via the interface.

The dashed red line shows input data flow to the backup interface. The dashed line stops at the interface because the interface does not send the data to buffering unless the interface is in the primary state. If the backup interface were to transition to the primary state for any reason, the interface would send the input data to buffering. Buffering would then write the data to all of the Historian Servers in the collective via n-way buffering.

The dashed blue line shows output data flow to the backup interface. The dashed line stops at the interface because the interface does not send the data to the data source unless the interface is in the primary state. If the backup interface were to transition to the primary state for any reason, the interface would send the output data to the data source.

In the event the Primary Historian Server becomes unavailable for any reason, the primary interface informs the backup interface that it has lost its connection to the Historian Server. The backup interface will transition to the primary state because its status is better than the current primary interface. If for example the entire Business Network were to go off line so that both copies of the interface lost their connection to their respective Historian Servers. The primary interface would remain primary because the backup interface's current status is the same, not better. In this case output data would no longer flow to the data source because there is no way for any of the interfaces to get the exception data.

Steady State Operation

Steady state operation is considered the normal operating condition. In this state, the primary interface is actively collecting data and sending its data to Historian. The primary interface is also updating its heartbeat point, monitoring the heartbeat point for the backup interface, and checking the active ID point every failover update interval. The backup interface is actively collecting and queuing data but not sending that data to Historian. It too is updating its heartbeat point, monitoring the heartbeat point for the primary interface, and checking the active ID point every failover update interval. As long as the heartbeat point for the primary interface indicates that it is operating properly and the active ID point has not changed, the backup interface will continue in this mode of operation.

The interaction of the control points is fundamental to failover. The discussion that follows only refers to the data written to the control points on the data source. However, every value written to the control points on the data source is echoed to the control tags on the Historian Server. Updating of the control tags is assumed to take place unless communication with the Historian Server is interrupted. The updates to the Historian Server will be buffered by bufserv or BufSS in this case.

Each interface participating in the failover solution will queue two failover intervals worth of data to prevent any data loss. When a failover occurs, there may be a period of overlapping data for up to 2 intervals. The exact amount of overlap is determined by the timing and the cause of the failover and may be different every time. Using the default update interval of 1 second will result in overlapping data between 0 and 2 seconds. The no data loss claim is based on a single point of failure. If both interfaces have trouble collecting data for the same period of time, data will be lost during that time.

As mentioned above, each interface has its own heartbeat point. In normal operation, the value of the Heartbeat point on the data source is incremented by UniInt from 1 – 15 and then wraps around to a value of 1 again. UniInt increments the heartbeat point on the data source every failover update interval. The default failover update interval is 1 second. UniInt also reads the value of the heartbeat point for the other interface copy participating in failover every failover update interval. If the connection to the Historian Server is lost, the value of the heartbeat point will be incremented from 17 – 31 and then wrap around to a value of 17 again. Once the connection to the Historian Server is restored, the heartbeat values will revert back to the 1 – 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

During steady state, the **`ActiveID`** will equal the value of the failover ID of the primary interface. This value is set by UniInt when the interface enters the primary state and is not updated again by the primary interface until it shuts down gracefully. During shutdown, the primary interface will set the `ActiveID` to zero before shutting down. The backup interface has the ability to assume control as primary even if the current primary is not experiencing problem. This can be accomplished by setting the **`ActiveID`** control point on the data source to the **`ActiveID`** of the desired interface copy.

As previously mentioned the backup interface actively collects data but does not send its data to Historian. To eliminate any data loss during a failover, the backup interface queues data in memory for two failover update intervals. The data in the queue is continuously updated to contain the most recent data. Data older than two update intervals is discarded if the primary

interface is in a good status as determined by the backup. If the backup interface transitions to the primary, it will have data in its queue to send to Historian. This queued data is sent to Historian using the same function calls that would have been used had the interface been in a primary state when the function call was received from UniInt. If UniInt receives data without a timestamp, the primary copy uses the current Historian Time to timestamp data sent to Historian. Likewise, the backup copy timestamps data it receives without a timestamp with the current Historian Time before queuing its data. This preserves the accuracy of the timestamps.

Glossary

The following is a list of the terms used by this manual when discussing UniInt Failover.

Active ID Point

The **active ID point** is located on the data source and identifies which copy of the interface is primary. The primary interface active ID value is set by the **`/UFO_ID=n`** startup command-line parameter for the primary interface copy. The value of **`n`** must be a positive, non-zero integer. The value of the active ID point is referred to as the **`ActiveID`**.

Backup Interface

The **backup interface** is used to describe the interface copy which is not the primary interface.

Cold Failover

In a **cold failover** configuration, the backup interface is not able to load the list of Historian tags, but is able to monitor the status of the primary interface. When the primary interface stops collecting data for any reason, the backup interface will load the list of Historian tags, start collecting data, and send its data to Historian. This configuration will result in a loss of data for the time it takes for the backup interface to load the tag list.

Data Source

The term **data source** is used as the generic term. The data source refers to the software/hardware where data originates that is to be written to Historian. The data source is also the destination of data output from Historian. Some examples of data sources are DCSs, PLCs, software systems using APIs, OPC servers, etc.

Failover ID

In a failover interface installation, each copy of the interface has its own unique positive integer identifier. This identifier is referred to as the **`failover ID`** and is specified with the interface startup parameter, **`/UFO_ID=n.`**

Failover Update Interval

The **failover update interval** determines the rate at which UniInt updates the heartbeat points, how long it takes for the interface to failover, and how much overlapping data may be sent to Historian. The optional **`/UFO_Interval`** startup parameter specifies the failover update interval for unsolicited failover control tags. Each copy of the interface may define the failover update interval as specified by the **`/UFO_Interval=i`**. However, both interface copies participating in failover must use the same value, **`i`**, specified by the **`/UFO_Interval=i`** parameter. The failover update interval, **`i`**, is specified in milli-seconds with the default being 1000 milli-seconds or 1 second. The valid range for the failover update interval is 50 – 20000 milli-seconds.

The **`/UFO_Interval`** startup parameter can only be used with unsolicited input interface failover control tags. If the interface supports scan based input data and the interface failover tags are defined as scan based input tags, the update interval is then defined by the scan class to which the control tags belong.

Heartbeat Point

The **heartbeat point** is used to indicate that the interface is functioning. This point is updated once every failover update interval. If the interface is connected to the Historian Server, the value of the heartbeat point will be incremented by UniInt from 1 – 15 and then wraps around to a value of 1 again. If the connection to the Historian Server is lost, the value of the heartbeat point will be incremented from 17 – 31 and then wrap around to a value of 17 again. Once the connection to the Historian Server is restored, the heartbeat values will revert back to the 1 – 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

Hot Failover

In a **hot failover** configuration, the interface copy that is in a backup role collects and queues data in parallel to the interface that is in the primary role. The interface in the backup role does not send the data collected to Historian. However, if a failover were to occur, the interface would immediately send its data to Historian. This mode of failover eliminates loss of data for a single point of failure.

Input Data

The term **input data** refers to data that originates on the data source and is collected by the interface and then written to the Historian Server.

Interface Control Points

An **interface control point** is a point on the data source that is used by UniInt to negotiate interface failover. The interface must be able to read the value of this point from the data source and write a value to this point on the data source. There are two types of interface control points: heartbeat points and active ID points. The value of these control points are echoed back to the Historian Server.

Interface Control Tags

The failover solution requires six **interface control tags** (Historian tags). When UniInt starts and loads the list of Historian tags, it will retrieve the information needed to be able to read from and write to the interface control points. If any of the six interface control tags are not found or fail to be loaded by the interface, the interface will log a message stating that the interface is not properly configured to participate in failover and then shutdown.

Interface Copy

The term **interface copy** refers to a single interface executable that is participating in a failover configuration.

Interface Instance

An **interface instance** is defined as an interface with a unique set of Historian tags determined by PointSource and interface ID **`(/id`** startup parameter). An interface instance may be redundant or standalone. When discussing a failover configuration, the interface instance includes both redundant copies of the interface since they have the same PointSource and interface ID.

Output Data

The term **output data** refers to data that originates in Historian or the failover copy of the interface itself, and written to the data source.

Primary Interface

When an interface is installed in a failover configuration, only one copy will be actively sending data from the data source to Historian and from Historian to the data source. This copy of the interface is referred to as the **primary** interface.

Point

For the discussion concerning failover, an entity that exists on the data source is referred to as a **point**. A point may also be referred to as a *control point*.

Tag

For the discussion concerning failover, an entity that exists on the Historian Server is referred to as a tag or Historian tag. A tag may also be referred to as a *control tag*.

Thrashing

Thrashing is defined as two interfaces continually alternating between the primary and backup roles. This condition must never be allowed to occur. A defective implementation of failover can lead thrashing.

Warm Failover

In a **warm failover** configuration, the backup interface does not actively collect data. The backup interface is able to load the list of Historian tags and is in a state ready to start collecting data as soon as the primary interface fails or stops collecting data for any reason. If the backup interface assumes the role of primary, it will start data collection and it will send its data to Historian. There may be some data disruption.

Failover Scenarios

There are a number of conditions that cause a failover as described below. The assumption is made in the following sections that the two interface copies are in a steady-state condition and a change in state forces a failover.

Primary Interface Shutdown

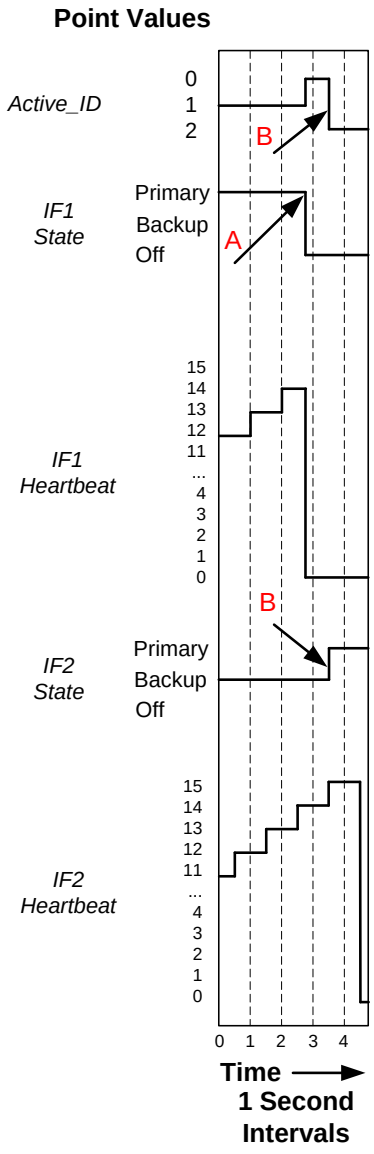
This scenario refers to a graceful shutdown of the primary interface. The chain of events is as follows.

- The primary interface is stopped.
- On exit, the primary interface writes a value of 0 to its heartbeat point and to the active ID point.
- The backup interface reads the **ActiveID**. As soon as it detects a value of 0, it assumes the role of the primary interface.
- The primary interface may stop just after the backup checked the value of the control points on the data source. It will then take an additional failover update interval before the backup realizes the primary has shut down. This will not result in a loss of data because the backup has two failover update intervals of data in its queue. When the backup transitions to the primary state, it sends its two intervals worth of queued data. This will result in between one and two failover update intervals worth of overlapping data (e.g., for the default failover update interval of 1 second, there will be up to 2 seconds of overlapping data).

Assuming Role of Primary Interface

- The interface in backup mode writes its failover ID to the active ID point.
- It waits two failover update intervals and then verifies that the **ActiveID** is the same as its failover ID.
- If the **ActiveID** is equal to the other copy's failover ID, the interface remains in backup mode. If the **ActiveID** is not the other copy's failover ID and it is not this copy's failover ID, the interface sets the **ActiveID** to its failover ID and waits another two failover update intervals. If the interface **ActiveID** remains the same, the backup interface sends its queued data to Historian and starts operating as the primary interface.

The timing chart in Figure 2 shows the primary interface gracefully shutting down and the backup interface failing over to assume the role as primary. Time increases from left to right according to the scale at the bottom of the figure. Every number along the horizontal axis represents one failover update interval. In the explanations that follow the charts, the failover update interval is the default of one second. Key events in time are denoted by the labeled arrows A, B, C, and D.



Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.5	IF2 reads the active ID point and IF1's heartbeat point. Both are good so IF2 discards data in its queue older than time 0.5.
T+2.8	Event A: IF1 shuts down gracefully, setting the active ID point and its heartbeat point to 0.
T+3.5	Event B: IF2 notices the active ID point is 0 and immediately transitions to primary, sending the data in its queue to Historian.
Overlap data from time 0.5 to 2.8	

Figure 2: Timing chart for when the primary interface gracefully shuts down and the backup interface assumes the role of primary.

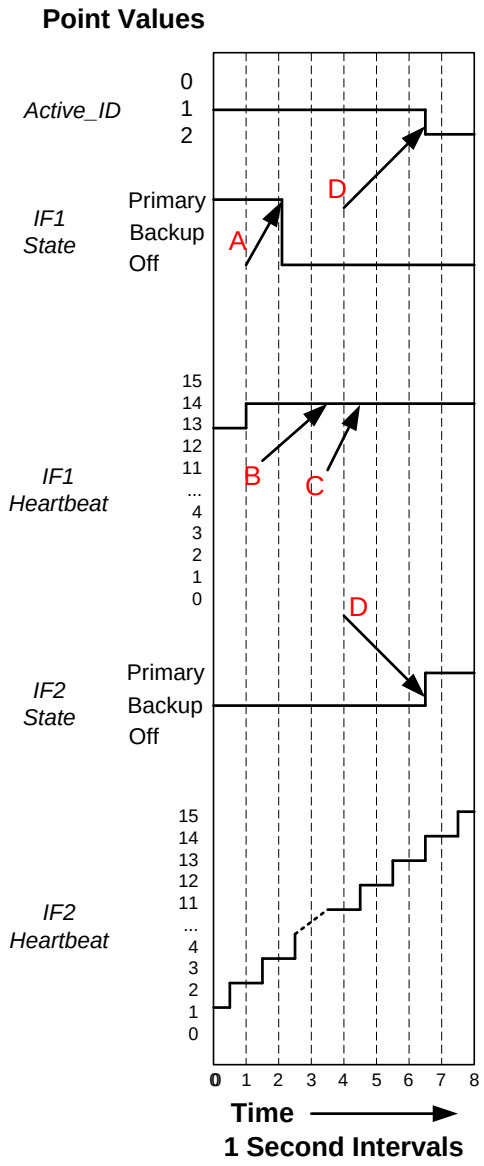
Primary Interface Stops Updating Heartbeat Point

There are a number of situations that can cause an interface to stop updating its heartbeat point. For example, the interface crashes, the interface is unable to write to the data source for any reason, or the interface is hung in a function call that does not return. The following scenario describes the steps taken by the backup copy of the interface to determine if it should assume the role as primary.

- Every failover update interval, the backup interface reads the value of the heartbeat point for the primary interface and stores the value internally.
- If the primary's heartbeat has not updated between two reads, the backup stops purging data older than two update intervals from its queue.
- If after two update intervals, the primary's heartbeat has not updated, the backup transitions into a temporary **`PrimaryStale`** state. This state allows the primary two additional update intervals of time to recover before the backup assumes control. If there was no recovery time for the primary, thrashing might result due to latency in the system architecture.
- Upon entering the **`PrimaryStale`** state, the backup stops discarding queued data.
- The backup interface remains in the **`PrimaryStale`** state for two failover update intervals waiting for the primary interface to come back to life. If the primary interface copy has still not updated its heartbeat point, the backup sends the data in its queue to Historian and transitions to the **`AssumingControl`** state.
- When the backup enters the **`AssumingControl`** state, it immediately writes its failover ID to the active ID point. At this point, the interface is now considered the primary copy. The interface remains in the **`AssumingControl`** state for two update intervals before transitioning to the **`PrimaryReady`** state.

Total failover time for this scenario occurs between three and five update intervals. The exact time it takes to failover depends on when the primary failed and when the backup read the control points. If the backup reads the primary's heartbeat just after the primary had updated its heartbeat and the primary then halts, the failover time will be closer to three intervals. If the backup reads the primary's heartbeat just before the primary updates its heartbeat, and then the primary fails just after updating its heartbeat the next interval, the failover time will be closer to five intervals. With the default update interval of 1 second, failover will occur between 3 and 5 seconds. The amount of overlapping data also depends on the timing, but will have a maximum of two intervals. Using the default update interval of 1 second may result in a maximum of 2 seconds worth of overlapping data.

Figure 3, below shows the failover timing chart related to the primary's failure to update its heartbeat point as discussed in the above scenario.



Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.1	Event A: IF1 stops updating its heartbeat point.
T+3.5	Event B: IF2 notices that IF1's heartbeat has not updated. IF2 will discard data older than Time 1.5.
T+4.5	Event C: IF2 notices that IF1 still has not updated its heartbeat. IF2 stops discarding old queued data.
T+6.5	Event D: IF1 still has not updated its heartbeat so IF2 transitions to Primary sending all data received from Time 1.5.
Two things should be apparent in this figure. The time it takes for the failover to occur is slightly less than four and one half seconds and the amount of overlapping data is just over one half second.	

Figure 3: Timing chart when the primary interface stops updating its heartbeat tag.

Manual Failover

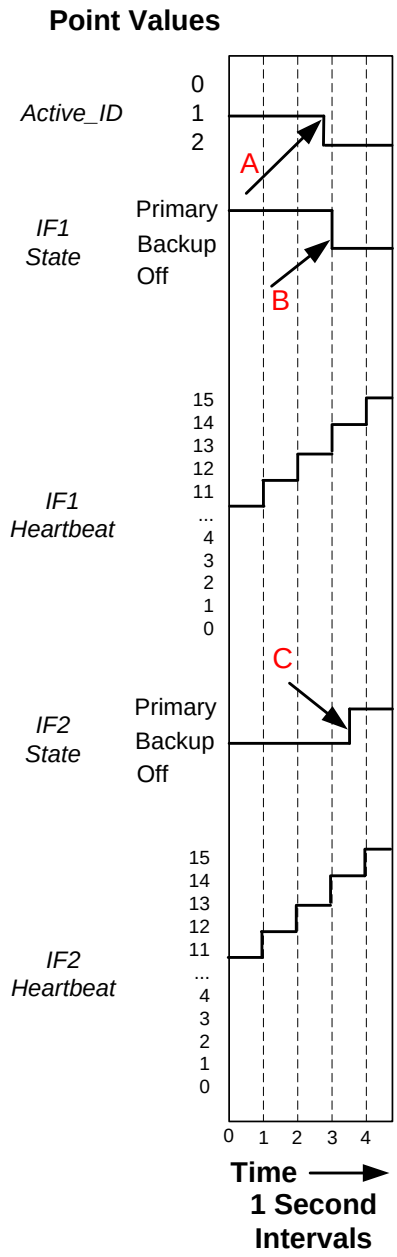
Initiating a manual failover requires that the backup interface be in a state equal to or better than the current primary copy of the interface. For example, if the current primary has a connection to the Historian Server and the backup does not, manual failover is not authorized. The backup interface will know the state of the primary by the value of the heartbeat point as explained in the next section.

Manual failover will result in overlapping data of between of up to two update intervals. Manual failover can be accomplished by changing the value of the active ID point on the data source from the current primary's failover ID to the current backup's failover ID. For example, if interface copy **`IF1`** is currently the primary and interface copy **`IF2`** is currently the backup, manual failover can be accomplished by changing the active ID point from a value of 1 to a value of 2 on the data source.

If the active ID point on the data source is changed, actions taken by the interface during this type of failover are as follows.

- The primary will notice that the value of the active ID point on the data source is equal to the failover ID of the other copy and immediately transition to the backup state.
- Similarly, the backup will notice that the value of the active ID point on the data source is now its failover ID and will immediately transition to the primary state.

Following is a depiction of the failover timing chart related to the manual failover scenario where the active ID point on the data source is changed.



Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.5	IF2 reads the active ID point and IF1's heartbeat point. Both are good, so IF2 discards data in its queue older than time 0.5.
T+2.8	Event A: the active ID point on the data source is changed from 1 to 2.
T+3	Event B: IF1 reads the active ID point and IF2's heartbeat point. IF2's heartbeat is good and the active ID has changed to a 2, so IF1 immediately transitions to backup.
T+3.5	Event C: IF2 notices that the active ID point on the data source is 2 and immediately transitions to primary. The data in its queue and any new data are sent to Historian.
Overlap data from time 0.5 to 3.	

Figure 4: Timing chart for a manual failover to the backup interface.

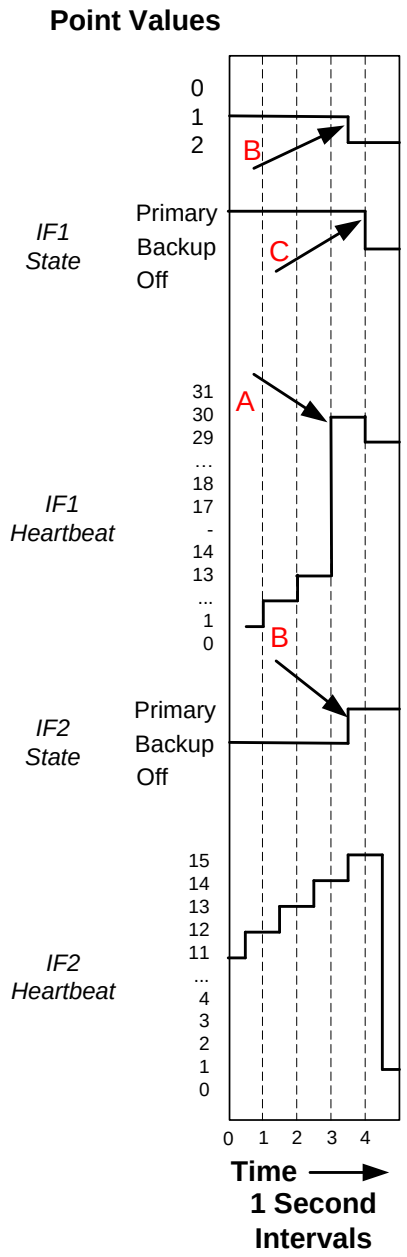
Primary Interface Loses Connection to Historian Server

Users may have criteria dependent upon continuous data flow to Historian. This failover prevents the disruption of data flow to Historian that may cause user dependent operations to fail when the connection to the Historian Server is lost. The following sequence of events is taken by the primary and backup interface if the primary interface loses its connection to the Historian Server.

- When the primary interface loses its connection to the Historian Server, it writes a value incrementing between 17 – 31 and then back to 17 to its heartbeat point. UniInt will send a signal to the state machine if any of the PI API functions return a value indicating that Historian is unavailable.
- The backup interface reads a value of 17 – 31 on the primary's heartbeat point during its next failover update interval.
- The backup transitions to a state indicating that the primary has lost its connection to the Historian Server (**`PrimaryNOPI`**).
- The backup no longer discards queued data.
- At the next failover update interval, the backup attempts to read the snapshot of the active ID tag from the Historian Server. If the read from the Historian Server is successful, the interface transitions to the `AssumingControl` state, updating the value of the active ID point on the data source to its failover ID. At this point, the backup has effectively taken over the role of primary.
- The old primary will notice that the value of the active ID point on the data source is equal to the failover ID of the other copy and immediately transition to the backup role. The newly designated backup interface begins to queue its data.

This scenario may result in up to 2 seconds of overlapping data depending on the timing. The overlapping or duplicate data will not appear in Historian until the interface that lost its connection to Historian reestablishes its communication to Historian and bufserv sends the buffered data.

Following is a failover timing chart of the situation where the primary interface loses its connection with the Historian Server.



Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.5	IF2 reads the active ID point and IF1's heartbeat point. Both are good, so IF2 discards data older than time 0.5.
T+3	Event A: IF1 updates its heartbeat to 30 because it has lost its connection to the Historian Server. When the connection to Historian is lost, the heartbeat for the interface increments between 17-31 and then back to 17. In this example the primary interface copy heartbeat was at 13 for IF1, so the next heart beat will be 30 if the connection to Historian is lost.
T+3.5	Event B: IF2 reads the active ID point and IF1's heartbeat point. IF2 still has a good connection to Historian, so it transitions to primary and sets the active ID point to 2.
T+4	Event C: IF1 reads the active ID point and notices the change to a 2. IF1 immediately transitions to backup.
Overlap data from time 0.5 to 4	

Figure 5: Timing chart for failover when the primary interface loses its connection with the Historian Server.

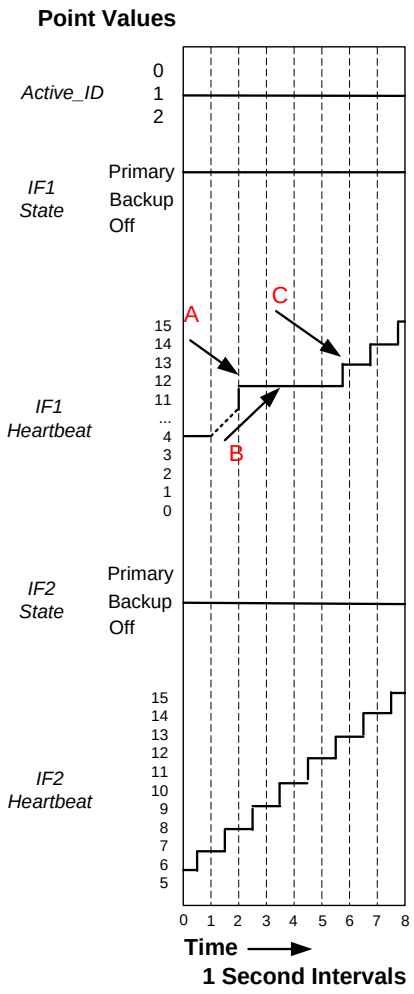
Primary Stops Updating Heartbeat Point for less than 5 Intervals

This situation might occur if the primary interface copy fails to update its heartbeat point within four failover intervals due to the interface being "stuck" somewhere in the code. In this scenario, the primary copy does come back to life and updates its heartbeat before the backup has assumed control. Moreover, it is assumed that the primary has missed scans for data during the time it failed to update its heartbeat. For this reason, the backup will send the data it had queued during the time frame where the primary stopped updating its heartbeat point. This results in two periods of overlapping data. The first period covers the time when the primary initially hangs and the second period covers the interval when the primary comes alive again. The first overlap is the same as if the primary never came back to life. The additional overlap occurs between the times when the primary came back and before the backup noticed.

- Every failover update interval, the backup interface reads the value of the heartbeat point for the primary interface and remembers the value internally.
- If on the third update interval the value of the primary's heartbeat point has not changed, the backup transitions into a temporary **`PrimaryStale`** state. This state allows the primary two additional failover update intervals of time to recover before the backup assumes control. If there was no recovery time for the primary, thrashing might result due to latency in the system architecture.
- Upon entering the **`PrimaryStale`** state, the backup stops discarding queued data.
- The backup interface remains in the **`PrimaryStale`** state for two failover update intervals waiting for the primary interface to come back to life.
- Before two failover update intervals elapse, the primary resumes normal operation and updates its heartbeat point.
- At the next failover update interval, the backup will see that the primary has resumed updating its heartbeat point and will transition back to the backup role.
- When the backup exits the **`PrimaryStale`** state, it will send its queued data to the Historian Server.

This scenario results in no loss of data even if the primary interface hangs between two and five intervals. If the primary hangs for less than two intervals, the backup never transitions to the `PrimaryStale` state and therefore never sends any queued data. This may result in missing scans for less than two intervals at a time.

A depiction of the timing chart for a scenario where the primary interface stops updating its heartbeat point for less than five intervals is shown in Figure 6, below.



Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.25	Event A: IF1 stops updating its heartbeat point for any reason.
T+2.5	IF2 reads the active ID point and IF1's heartbeat point. Both are good so IF2 discards data older than time 0.5.
T+3.5	Event B: IF2 notices that IF1's heartbeat point has not updated. IF2 discards data older than Time 1.5.
T+4.5	IF2 notices that IF1 still has not updated its heartbeat point. IF2 stops discarding queued data.
T+5.5	IF2 notices that IF1 still has not updated its heartbeat point.
T+5.75	Event C: IF1 resumes operation. The active ID point is equal to IF1's ID, so IF1 updates its heartbeat point and continues operating normally.
T+6.5	IF2 notices IF1 has updated its heartbeat point. IF2 will send the data it has in its queue and then return to normal backup operation.
IF1 was not collecting data from time 2.25 to time 5.75. IF2 will send the data it received from time 1.5 to time 6.5. This will lead to two periods of overlapping data: from time 1.5 to time 2.25 and from time 5.75 to time 6.5.	

Figure 6: Timing chart for failover when the primary interface stops updating its heartbeat point for less than five intervals.

Interfaces Connect to Data Source at Approximately the Same Time

There may be a chance where both copies of the interface connect to the data source at approximately the same time and contend to become the primary interface. For example, both interfaces are running and then the data source is started, or both copies of the interface are started at the same time. If the data source is unavailable and the interface is running, attempts to update the failover control points will return an error. If the interface receives errors reading from or writing to the data source, it will transition to the backup role. At start-up, the interface starts in the backup role. It initially checks the value of the active ID point and the heartbeat point of the other copy of the interface. The value of the active ID point will be one of three values: the failover ID for this copy of the interface, the other copy's failover ID, or something else. These three scenarios are discussed below.

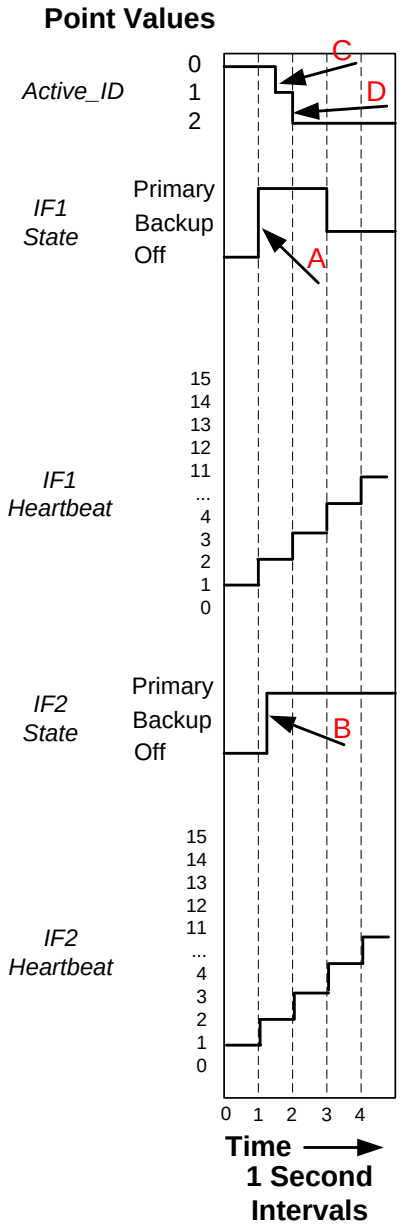
- If the **ActiveID** equals the failover ID for this copy of the interface, the interface transitions to the primary role and starts sending data to Historian. In this situation, it will take two failover update intervals for data to show up in Historian. This is because the interface actually transitions to the **AssumingControl** state for two intervals and queues its data. No data is lost during this two failover update interval start-up delay. The delay is used to prevent contention between the two interfaces trying to assume the role of primary at the same time.

- If the **ActiveID** equals the failover ID for the other copy of the interface, the interface stays in the backup role and starts monitoring the heartbeat of the other copy. For example, if both copies of the interface are down and then one copy is started and the active ID point is equal to the other copy's failover ID, it will take four update intervals for its data to show up in PI. This is because the interface assumes the backup role at startup and waits to verify the other copy's heartbeat. If the other copy's heartbeat point is not being updated, this copy will go through the process of assuming the role of primary. All data collected by the interface during this four interval start-up delay will be sent to PI.

- If the **ActiveID** equals something other than this copy's or the other copy's failover ID, the interface transitions to the `AssumingControl` state. Again, the interface will stay in this state for up to two failover update intervals to avoid contention. For example, if interface **IF1** reads the **ActiveID**, then interface **IF2** reads the **ActiveID** before interface **IF1's** update takes place. Interface **IF1** will claim primary by setting the `ActiveID` to its failover ID. Interface **IF2** will claim primary by setting the **ActiveID** to its failover ID. Then both copies wait one failover update interval and read the value of the active ID point. If the **ActiveID** has changed to the other copy's failover ID, the interface will transition to the backup role. If the **ActiveID** is still equal to this copy's failover ID, the interface will wait one more interval. If, after two intervals, the `ActiveID` still equals this copy's failover ID, the interface will transition into the primary role and send the data it has queued during this process.

  In any of the situations where a backup assumes control by updating the active ID point, the update takes place immediately after reading the point. The two interval delay resolves the collision problem and will only be an issue if both interfaces read the value of the active ID point before the update reaches the data source. If the latency for a write is more than two failover update intervals, then the failover update interval should be increased. This algorithm results in the last interface to make the update to the active ID point remaining the primary interface.

Refer to Figure 7,below which describes the failover action when both interface copies start at the same time and the latency for an update to the data source is equal to one half of an update interval.



Time	Action
T+0	Neither interface is running
T+1	Event A: IF1 starts and notices the active ID point is 0 so it transitions to active and writes a 1 to the active ID point.
T+1.25	Event B: IF2 starts and notices the active ID point is 0 so it transitions to active and writes a 2 to the active ID.
T+1.5	Event C: the update from IF1 reaches the data source
T+2	Event D: IF1 reads a 1 for the active ID point. Immediately after this, the update from IF2 reaches the data source.
T+2.5	IF2 reads a 2 for the active ID point and remains in the active mode.
T+3	IF1 reads a 2 for the active ID and immediately transitions to the backup role.
The last interface to update the active ID point is the interface that will remain in the primary role.	

Figure 7: Timing chart for both interfaces starting at the same time.

Maximum Data Overlap

Figure 8 shows the timing chart for a scenario that allows the maximum overlapping of data during a failover.

Error! Objects cannot be created from editing field codes.

Time	Action
T+0	Both interfaces are running with IF1 in the primary role.
T+2.1	IF2 reads the ActiveID and IF1's heartbeat. Both are good, so IF2 discards data older than time 0.1.
T+2.9	Event A, IF1's stops for any reason just before updating its heartbeat.
T+3.1	Event B, IF2 notices that IF1's heartbeat has not updated. IF2 will discard data older than Time 1.1.
T+4.1	IF2 notices that IF1 still has not updated its heartbeat. IF2 stops discarding old queued data.
T+5.1	IF2 notices that IF1 still has not updated its heartbeat.
T+6.1	Event C, IF1 still has not updated its heartbeat so IF2 transitions to Primary sending all data received from Time 1.1 and sets the ActiveID to 2.
The time it takes for the failover to occur is slightly more than three seconds and the amount of overlapping data is almost two seconds from time 1.1 to time 2.9.	

Figure 8: Timing Chart for the Minimum Failover Time with Maximum Overlap.

Error and Informational Messages

Error messages that are written to the message log are prepended by a string **NameID**. **Name** is a non-configurable identifier that is no longer than 9 characters. **ID** is a configurable identifier that is no longer than 9 characters and is specified using the **/id** parameter on the startup command-line.

Message Logs

The location of the message log depends upon the platform on which the interface is running.

Messages on Windows

Error and informational messages are written to the **pipc.log** file. The location of the **pipc.log** file is determined by the **PIHOME** entry in the **pipc.ini** file. The **pipc.ini** file should always be in the **%windir%** directory. For example, if the **PIHOME** entry is:

```
C:\Program Files\PIPC
```

then the **pipc.log** file will be located in the **c:\Program Files\PIPC\dat** directory.

For version 1.3 and greater of the PI API, a process called **pilogsrv** is installed to run automatically as a service. After the **pipc.log** file exceeds a user-defined maximum size, the **pilogsrv** process renames the **pipc.log** file to **pipcxxxx.log**, where **xxxx** ranges from 0000 to the maximum number of allowed log files. Both the maximum file size and the maximum number of allowed log files are configured in the **pipc.ini** file. Configuration of the **pilogsrv** process is discussed in detail in the *PI-API Installation Instructions* manual.

System Errors and PI Errors

System errors are associated with positive error numbers. Errors related to PI are associated with negative error numbers. Interface-specific errors are discussed in the interface-specific documentation.

Error Descriptions on Windows

On Windows, descriptions of system and PI errors can be obtained with the pidiag utility:

```
\PI\adm\pidiag -e error_number
```

Index

Attribute Point
CompDev, 19
CompDevPrecent, 19
CompMax, 19
CompMin, 19
Compressing, 18
DigitalSet, 21
ExcDev, 18
ExcDevPercent, 18
Exception Specification, 17
ExcMax, 18
ExcMin, 18
ExDesc, 21
Extended Descriptor, 21
Location4, 19
PointType, 17
Scan, 19
SourceTag, 20
Tag, 16
TypicalValue, 19
Bufserv
Definition of, 2
Shutdown events, 20
Command-line Parameters, 3
Compatibility mode, UniInt 2.x, 31
Compressing attribute, 18
CompDev, 19
CompDevPercent, 19
CompMax, 19
CompMin, 19
Connection establishment, interface, 29
DigitalSet attribute, 21
Error Returns
Interfaces, 99
For Windows, 99
ExcDevPercent, 17
Exception Specification attribute, 18
ExcDev, 18
ExcDevPercent, 18
ExcMax, 18
ExcMin, 18
ExDesc, 21
Exit handler, 30
Extended Descriptor attribute, 21, 22
Failover-UniInt, 82
Failover Scenarios, 85
Glossary, 82
Introduction, 73
Steady State Operation, 81
I/O Rates
Point, 25
Input point, 22
Definition of, 15
Scan-based, 22
Trigger based, 22
Unsolicited input, 23
Interface Startup Parameters, 3
Known Servers Table, 28
Library
PI API, 29
PI SDK, 29
Location4, 19, 22
Performance point, 27
Log Files
Purging, 99
Modbus, 9
Output point, 24
Definition of, 15
Parameters

Command-line, 3
Performance point, 26
Performance summary, 27
PI errors, 99
PI point
Definition of, 15
Interface specific points, 15
PI SDK
Support of interfaces, 28
pilogsrv, 99
Installation, 99
Point types, 17
Point updates, 30
PointSource attribute
Command line parameters, 11
PointType. *See* Point types
Scan attribute, 19
Scan classes, 19
Shutdown attribute
Bufserv, 20
SourceTag attribute, 20
Swinging-door algorithm, 19
System errors, 99
Tag
Definition of, 16
Time
UniInt 3.x, 31
Timestamp
Extended PI API mode, 31
UniInt 2.x Compatibility mode, 31
Trigger point, 24
TypicalValue, 19
UniInt
2.x Compatibility mode, 31
definition of, 1
Universal Coordinated Time. *See* UTC time
Universal Interface, 1
UTC time, 32
Zero and Span attributes, 19